UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA
DIVISIÓN DE ESTUDIOS DE POSGRADO

DESARROLLO DE UN PROGRAMA DE SIMULACIÓN DE UN MODELO

CINEMÁTICO DE UN ROBOT ARTICULADO DE 6 GRADOS DE

LIBERTAD CRS A465 DEL ITESCA

Tesis que para obtener el Título de
Maestro en Ingeniería
(Área de Mecánica)
presenta:
I. E. Javier Jacobo Peña

Asesor del trabajo de Tesis:
M. I. Leopoldo González González

MÉXICO, D. F. CIUDAD UNIVERSITARIA 2006

Dedicatoria

A mis hijas Zurisadai, Kim Verónica y Dan Ahira, a mi esposa Verónica y a mi madre María de Jesús que son mi razón de ser.

A fin de alcanzar una meta a la que nunca has llegado, tendrás que hacer cosas que nunca has hecho. Elder Richard G. Scott. Liahona, Julio de 1990.

Every individual should have a purpose in life, which is worthy of intense effort – and constantly work toward the definite goal ahead. Roderick Stevens

Agradecimientos

Agradezco infinitamente a mi Dios por darme la oportunidad de venir a esta tierra y por poner todos los medios necesarios para mi desarrollo en el camino eterno de la vida.

Agradezco también a mi familia; incluyendo a mis hijas, a mi esposa, a mi madre y a mis hermanos por su apoyo incondicional en el transcurso de mi vida y mi preparación.

Agradezco a mis maestros que dieron lo mejor de sí para compartir sus conocimientos y alentarnos a ir más allá de lo visto.

Agradezco especialmente a mi asesor de Tesis por su apoyo y tiempo dedicado en el desarrollo de este trabajo.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

Resumen

La simulación por computadora de los movimientos de cualquier máquina o multicuerpo rígido es hoy en día una técnica muy útil para la visualización y análisis del desempeño del mismo; sirve, por ejemplo para prever cualquier posible problema de interferencia entre las partes mecánicas del sistema o la simple visualización del proceso en sí. La utilización de gráficos con sólidos en tercera dimensión, sombreados y con animación de los movimientos, hace posible una mejor visualización de las escenas al ofrecer un mejor realismo. OpenGL®, es un estándar en el manejo de gráficos por computadora ampliamente aceptado en el mundo entero; el cual facilita en gran medida el desarrollo de aplicaciones de este tipo.

El presente trabajo presenta el desarrollo de un programa para computadora para la simulación de los movimientos y operación manual del robot articulado de 6 grados de libertad modelo CRS A465; utilizando una librería tipo OpenGL® compatible con el lenguaje de programación Visual Basic. Para la modelación de la cinemática de dicho robot, se aprovechan las características y ventajas del álgebra de quaterniones con respecto al resto de los métodos disponibles.

La finalidad del desarrollo de este trabajo es proporcionar a los alumnos del Instituto Tecnológico Superior de Cajeme (ITESCA) una herramienta alterna para el aprendizaje de la utilización del robot CRS A465, desde la comodidad de una computadora personal; con las ventajas que ello conlleve; es decir, sin poner en riezgo la integridad tanto del usuario como del mismo robot o su ambiente de trabajo; así como la disponibilidad del simulador en cualquier computadora personal, por lo que cualquier alumno pudiera tener acceso a él cuando sea requerido.

Abstract

Computer aided simulation of a machine or a multipart rigid body kinematic is today a useful practice for visualization and analysis of its performance; it is used, for previewing possible interference problem between mechanical components or simple process visualizing. Utilizing graphics with 3D shadowed, animated solids improves visualization of the process scenes making them more realistic. OpenGL® is a world wide accepted computer graphics standard that easies the development of graphics applications.

This work presents the development of a computer application for simulating the kinematics and manual operation of a CRS A465 6 DOF articulated robot; using an OpenGL® type library for the Visual Basic programming language. Due to its advantages against other methods, quaternions algebra is used for kinematics modeling of this robot.

The main objective of this work is to give to the ITESCA (Instituto Tecnologico Superior de Cajeme) an alternate tool for learning the operation of the CRS A465 robot, all from a personal computer; with all the advantages that this carries; and without risking the robot or its work area or the user safety; also offering the availability of the simulator on any personal computer, so any student could have access to the system whenever it is required.

Introducción

México, siendo un país en vías de desarrollo, dentro de un marco de competitividad global se encuentra en la etapa donde todavía es importante la adopción de tecnologías desarrolladas en el extranjero para lograr crecimiento económico. El gobierno mexicano dice estar comprometido a lograr que la competitividad se convierta en el principal motor de nuestra economía [CECIC, 2002]. Para lograr ser competitivos es necesario no solamente adquirir tecnologías del extranjero y aprender a utilizarlas; sino también participando activamente en la innovación tecnológica al generar, apoyar y formar parte de programas de investigación científica y tecnológica.

El Instituto Tecnológico Superior de Cajeme (ITESCA) conciente de su compromiso con la sociedad y el país; ha venido impulsando la divulgación de conocimientos científicos y tecnológicos al invertir en la creación de laboratorios de alta tecnología y en conjunto con la Universidad Nacional Autónoma de México (UNAM) han establecido el convenio de impartir en las instalaciones del ITESCA el programa de Maestría en Ingeniería Mecánica con acentuación en diseño mecánico incluyendo el área de la robótica y CAD-CAM; poniendo así al alcance de profesionales locales la formación académica adecuada para participar en las líneas de investigación relacionadas con dichas áreas.

Dentro de este contexto, y dentro de un marco de autoequipamiento de la institución y el desarrollo de las disciplinas de investigación y desarrollo tecnológico, el presente trabajo pretende presentar una alternativa de operación para un Robot CRS A465 localizado en las instalaciones del ITESCA al desarrollar un programa de simulación virtual en tres dimensiones de un ambiente de trabajo real

para ofrecer al alumnado la oportunidad de aprender a utilizar dicho Robot desde cualquier computadora personal sin necesidad de operar físicamente el mismo; evitando así posibles riesgos tanto físicos del aparato como de la integridad del mismo estudiante. Ofreciendo además la facilidad de tener dicho programa de simulación instalado en cualquier otra localización y no necesariamente en las inmediaciones del Robot en sí.

El objetivo de este trabajo es desarrollar un programa en el lenguaje de programación Visual Basic versión 6.0 utilizando librerías OpenGL® para simular la cinemática de un Robot CRS A465 modelado matemáticamente por medio de quaterniones.

En el capítulo 1 se presentan tres de los programas de simulación para robots que se encuentran disponibles en el mundo.

En el capítulo 2 se presenta como marco teórico una explicación de los diferentes tipos de proyecciones que se usan para representar objetos tridimensionales en planos.

En el capítulo 3 se da una explicación de un método de desarrollo de la proyección paralela ortogonal para representar cuerpos en un espacio tridimensional sobre un plano de proyección visto desde un punto de vista ubicado arbitrariamente en el espacio. Se explica además una forma para representar cuerpos sólidos sombreados por el efecto de una iluminación de la escena por medio de una fuente de luz plana y se da un ejemplo de aplicación.

En el capítulo 4 se explica la utilización de una librería llamada OpenGL®, para la generación de imágenes de alta calidad para representación de objetos tridimensionales.

En el capítulo 5 se da una reseña de la cinemática de cuerpos en el espacio y se muestran algunos de los métodos comúnmente utilizados para representar el modelo matemático de la cinemática de multicuerpos rígidos.

En el capítulo 6 se da una explicación de la generación de un modelo matemático por medio de quaterniones para representar la cinemática de multicuerpos rígidos y su aplicación práctica para generar el modelo matemático de la cinemática del robot CRS A465.

En el capítulo 7 se explica el desarrollo del programa para simulación de la cinemática del robot CRS A465; cuyo modelo matemático se explicó en el capítulo 6 y la utilización de la librería OpenGL® para su graficación.

Antecedentes

Debido a muchos factores que van desde económicos, sociales, culturales y políticos, en México como en otros países en vías de desarrollo se ha venido presentando e incrementando un atraso tecnológico con respecto a los países altamente desarrollados; tales como Estados Unidos, Japón, Suiza y Alemania entre otros. En un artículo publicado por el Instituto Tecnológico Superior de Cajeme (ITESCA) en su página de Internet se expone lo siguiente:

"Una característica fundamental actual de México en lo que respecta a su nivel y calidad en el desarrollo de la ciencia y tecnología, es la existencia de un gran atraso histórico en relación con los países considerados de primer nivel". [ITESCA]

"México se caracteriza por un significativo atraso en su capacidad científica y tecnológica, comparado con los países considerados de primer nivel, presenta también, un crecimiento muy pobre en lo que respecta al desarrollo de la ciencia y tecnología, y, además, se está incrementando el gran distanciamiento tecnológico respecto a estos países, lo que implica entre otras cosas una fuerte dependencia económica". [ITESCA]

A continuación se enlistan algunos de los muchos factores que propician tal atraso:

- ✓ *Un número limitado de programas de formación de recursos de alto nivel.*
- ✓ *Reducida infraestructura para el impulso a la investigación científica.*
- ✓ *Políticas gubernamentales no propicias para el fomento de la investigación.*

- ✓ *Mínima incorporación del desarrollo tecnológico a los procesos productivos.*
- ✓ *Bajos subsidios a las actividades de investigación.*
- ✓ *Limitado impulso a las actividades de investigación por parte de empresarios.*
- ✓ *Inexistencia (o un pequeño número) de redes de investigadores.*

La industria mexicana se ha distinguido por su bajo nivel de utilización de automatización y tecnología al continuar utilizando a través de los años procesos puramente artesanales o semiautomáticos para producción. El ITESCA dice: *"Las características fundamentales del desarrollo de la tecnología en México, son apreciables en la forma en que la Industria Mexicana realiza sus procesos, donde imperan mas bien procesos artesanales o de muy baja automatización, lo cual se traduce en resultados que caracterizan a México como un país dependiente de tecnología, y una nación manufacturera e importadora".* [ITESCA]

En materia de robótica como ejemplo de un tipo de tecnología aplicada a los procesos productivos de la industria mexicana se aprecia un atraso enorme con respecto a las industrias de los países desarrollados. En México es poca la aplicación de la robótica para la automatización de proyectos como también lo dice el ITESCA: *"en materia de la robotización de la industria, México presenta un atraso de 30 años, por lo que para que nuestro país alcance una posición de competencia en el desarrollo industrial internacional para el 2020, requerirá aproximadamente 60 mil robots instalados (actualmente sólo contamos con seis mil de estos robots)".* [ITESCA]

Para lograr que México salga de su atraso económico es necesario que se modifique la forma en que se enfrenta el desarrollo económico del país. En el Programa Regional de Competitividad Sistémica,

un documento publicado por el gobierno del estado de Coahuila, en concordancia con el Programa de Desarrollo Empresarial 2001-2006 planteado por el Presidente Vicente Fox, en donde dice:

"Estamos decididos a que la ***competitividad*** *se convierta en el* ***eje central de la nueva política económica*** *que conduzca a las empresas mexicanas por el camino del crecimiento sustentable".* [CECIC, 2002]

En la presentación del Programa Regional de Competitividad Sistémica mencionado, se aclara el enfoque de dicho programa, diciendo:

"La propuesta que plantea el Programa define no sólo la Estrategia de Competitividad Internacional a seguir por las principales industrias coahuilenses; también representa un Programa de Acción para el desarrollo de la competitividad sistémica de nuestras empresas, distritos productivos y polos regionales, en donde el empresariado y sus organizaciones deberán llevar el liderazgo de su instrumentación. El objetivo fundamental es proyectar decididamente el rumbo competitivo y sustentable de Coahuila en su inserción exitosa a la economía globalizada". [CECIC, 2002]

Dentro de un contexto global de mercados, México, siendo parte integrante de algunos tratados internacionales de comercio; tal como el Tratado de Libre Comercio de América del Norte (NAFTA, por sus siglas en inglés), integrado por Canadá, Estados Unidos y México, entre otros de igual importancia, los diferentes sectores de la economía nacional, tales como sector productivo, educativo y de gobierno, por ejemplo; deben entender que el concepto nuevo de la competencia de mercados se rige bajo normas diferentes a las, hasta ahora conocidas.

México es un país con problemas de competitividad en el ámbito global; tal como se refleja en el Índice de Crecimiento Competitivo ICC (GCI por sus siglas en inglés), desarrollado por Jeffrey Sachs y McArthur en el Reporte Global de Competitividad 2004-2005 [Comrade]. El ICC está dirigido específicamente a medir la habilidad de las economías del mundo para alcanzar un crecimiento económico sostenido sobre un mediano o largo plazo. El cual da una idea de cuáles serían los factores que determinan el crecimiento a mediano plazo una economía. Está compuesto de tres pilares, los cuales son ampliamente aceptados como críticos al crecimiento económico: la calidad en el ambiente macroeconómico, el estado de las instituciones públicas de un país y, dado la creciente importancia de la tecnología en el proceso de crecimiento, la preparación tecnológica del país. Sachs y McArthur fuertemente hacen énfasis en que el papel de la tecnología en el proceso de crecimiento difiere de país a país, dependiendo particularmente en su situación particular de desarrollo. Es ampliamente entendido que la innovación tecnológica es relativamente más importante para el crecimiento de los países que se encuentran cerca de la frontera tecnológica. La innovación sería la clave en Suecia, no así la adopción de tecnologías del extranjero, o el tipo de transferencia tecnológica frecuentemente asociada con la inversión extranjera directa sería más importante en un país como la República Checa. Por esta razón, al estimar el ICC, las economías se separan en dos grupos: las economías del núcleo; que son aquellas donde la innovación tecnológica es crítica para el crecimiento, y las economías fuera del núcleo; aquellas que todavía pueden crecer al adoptar tecnologías desarrolladas en el extranjero. Este índice dice que México ocupa el lugar número 48 de una lista de 104 países (Tabla 1). En la Tabla 2, se muestra la posición que guarda México frente a sus competidores y socios comerciales dentro del Tratado de Libre Comercio de América del Norte.

Tabla 1. Growth Competitiveness Index rankings and 2003 comparisons

Country	GCI 2004 rank	GCI 2004 score	GCI 2003 rank*
Finland	1	5.95	1
United States	2	5.82	2
Sweden	3	5.72	3
Taiwan	4	5.69	5
Denmark	5	5.66	4
Norway	6	5.56	9
Singapore	7	5.56	6
Switzerland	8	5.49	7
Japan	9	5.48	11
Iceland	10	5.44	8
United Kingdom	11	5.30	15
Netherlands	12	5.30	12
Germany	13	5.28	13
Australia	14	5.25	10
Canada	15	5.23	16
United Arab Emirates	16	5. 21	—
Austria	17	5.20	17
New Zealand	18	5.18	14
Israel	19	5.09	20
Estonia	20	5.08	22
Hong Kong SAR	21	5.06	24
Chile	22	5.01	28
Spain	23	5.00	23
Portugal	24	4.96	25
Belgium	25	4.95	27
Luxembourg	26	4.95	21
France	27	4.92	26
Bahrain	28	4.91	—
Korea	29	4.90	18
Ireland	30	4.90	30
Malaysia	31	4.88	29
Malta	32	4.79	19
Slovenia	33	4.75	31
Thailand	34	4.58	32
Jordan	35	4.58	34
Lithuania	36	4.57	40

Country	GCI 2004 rank	GCI 2004 score	GCI 2003 rank*
El Salvador	53	4.10	48
Uruguay	54	4.08	50
India	55	4.07	56
Morocco	56	4.06	61
Brazil	57	4.05	54
Panama	58	4.01	59
Bulgaria	59	3.98	64
Poland	60	3.98	45
Croatia	61	3.94	53
Egypt	62	3.88	58
Romania	63	3.86	75
Colombia	64	3.84	63
Jamaica	65	3.82	67
Turkey	66	3.82	65
Peru	67	3.78	57
Ghana	68	3.78	71
Indonesia	69	3.72	72
Russian Federation	70	3.68	70
Algeria	71	3.67	74
Dominican Republic	72	3.63	62
Sri Lanka	73	3.57	68
Argentina	74	3.54	78
Gambia	75	3.52	55
Philippines	76	3.51	66
Vietnam	77	3.47	60
Kenya	78	3.45	83
Uganda	79	3.41	80
Guatemala	80	3.38	89
Bosnia and Hercegovina	81	3.38	—
Tanzania	82	3.38	69
Zambia	83	3.36	88
Macedonia, FYR	84	3.34	81
Venezuela	85	3.30	82
Ukraine	86	3.27	84
Malawi	87	3.24	76
Mali	88	3.24	99

Greece	37	4.56	35	Serbia and Montenegro	89	3.23	77
Cyprus	38	4.56	—	Ecuador	90	3.18	86
Hungary	39	4.56	33	Pakistan	91	3.17	73
Czech Republic	40	4.55	39	Mozambique	92	3.17	93
South Africa	41	4.53	42	Nigeria	93	3.16	87
Tunisia	42	4.51	38	Georgia	94	3.14	—
Slovak Republic	43	4.43	43	Nicaragua	95	3.12	90
Latvia	44	4.43	37	Madagascar	96	3.11	96
Botswana	45	4.30	36	Honduras	97	3.10	94
China	46	4.29	44	Bolivia	98	3.09	85
Italy	47	4.27	41	Zimbabwe	99	3.03	97
Mexico	48	4.17	47	Paraguay	100	2.99	95
Mauritius	49	4.14	46	Ethiopia	101	2.93	92
Costa Rica	50	4.12	51	Bangladesh	102	2.84	98
Trinidad and Tobago	51	4.12	49	Angola	103	2.72	100
Namibia	52	4.11	52	Chad	104	2.50	101

Fuente: Reporte Global de Competitividad 2004-2005. [FEM]

Tabla 2. Comparativo del crecimiento competitivo en América del Norte (lugar entre 104 países)

Índices/ País	México	Estados Unidos	Canadá
Índice Global ICC	48	2	15
Índice de Tecnología	48	1	13
Índice de Instituciones Públicas	59	21	18
Índice de Ambiente Macroeconómico	49	15	18

Fuente: Reporte Global de Competitividad 2004-2005. [FEM]

El Programa Regional de Competitividad Sistémica del Estado de Coahuila dice:

"En los últimos años, el fenómeno de la Globalización de los mercados y el aumento de la competencia ha obligado a que el concepto de competitividad de las regiones cobre vida en distintos estados del país. La competitividad se basa principalmente en la continua innovación tecnológica, que bajo un ambiente de gran incertidumbre, es facilitada en gran medida por la transmisión de conocimientos en el territorio. La principal fuente de desarrollo tecnológico de las regiones se ubica en la capacidad de asimilar en forma eficiente el creciente acervo tecnológico mundial. [CECIC, 2002]

Nótese que plantea que la "*competitividad se basa principalmente en la continua innovación tecnológica"*, la que aunque con "*gran incertidumbre, es facilitada en gran medida por la transmisión de conocimientos en el territorio"*; es decir por la educación y entrenamiento de la gente en el entendimiento y manejo de las tecnologías de punta, tal como continua diciendo:

"Es importante señalar que la región frontera cuenta con una Ventaja Comparativa muy importante y esta es la ubicación geográfica, sin embargo, esto no ha sido suficiente para que la Región desarrolle un sector competitivo. Es necesario invertir en tecnología y capacitar a los obreros para conseguir mano de obra calificada, ya que la que existe actualmente es de bajo valor añadido para la maquila de exportación es por eso que actualmente este sector económico presenta ciertos problemas como el que las empresas maquiladoras se trasladen a países con mano de obra calificada el caso de los países asiáticos. Esto provoca un problema para el Estado y más para un país como México que genera un porcentaje considerable en el valor agregado de un sector como la maquila de exportación". [CECIC, 2002]

Hasta ahora, lo que plantea el documento no es más que el despertar el interés de las empresas mexicanas a importar tecnología de punta y aprender a usarla; sin embargo, cambiando un poco el enfoque, también sugiere que, abriendo un poco más los ojos, establece que para ser más competitivos es necesario participar activamente en la innovación tecnológica:

"La globalización nos introduce a un nuevo juego: la hipercompetencia global en el mercado local, en la que existen nuevas reglas que implican la Ventaja Competitiva Sustentable, la eficiencia y el uso estratégico del capital intelectual; asimismo existen nuevos jugadores...

En la Nueva Economía Global y en medio de la hipercompetencia global en nuestro propio mercado local, sumando a ello la entrada de China a la OMC, la Ventaja Competitiva de México no está más en la mano de obra barata, ni en la explotación de los recursos naturales; tampoco reside solamente en su proximidad al mercado de Estados Unidos, sino en la capacidad para aprender e innovar *y potenciar los diversos sectores donde México presenta una Ventaja Competitiva Revelada, transformándola en una auténtica Ventaja Competitiva Sustentable y Sistémica (VCS*^{2}*)".* [CECIC, 2002]

Involucrando al sector educativo de nuestra nación, llamando la atención, plantea que se encaminen los esfuerzos a la innovación y al desarrollo de un *"Capital Intelectual"*; como lo presenta:

"El Capital Intelectual que es el nuevo factor de competitividad en la era del conocimiento y que requiere ir más allá del concepto tradicional de desarrollo científico y tecnológico, para enfocar éste en la capacidad creativa sistémica para promover la innovación en los diferentes campos, a través de un Sistema Nacional de Innovación apoyado en nuevos enfoques de educación (tanto formal como en la empresa) que enfaticen el "aprender a aprender", el "aprender a emprender" y el "aprender haciendo" en los propios procesos del trabajo productivo...

El desarrollo del capital intelectual y de innovación implica, en primer lugar, contar con una sólida infraestructura educativa, a fin de generar el capital humano que demanda la hipercompetitividad global. Este capital humano es vital para las empresas, tanto por lo que respecta a la formación de trabajadores del conocimiento. La infraestructura tecnológica comprende otros factores, como la adecuada inversión en capital físico y maquinaria, lo mismo que la base institucional que favorece la

existencia de un sistema de innovación, tanto en investigación y desarrollo, como en procesos de producción y comercialización". [CECIC, 2002]

Es necesario entonces que las instituciones educativas participen en la formación de individuos comprometidos con el desarrollo tecnológico de México, generando líneas de investigación que ayuden a tal propósito de modo que en forma conjunta participen en las diferentes áreas del conocimiento para propiciar dicho desarrollo. Por su parte el ITESCA asegura que:

"Para que México sea competitivo industrialmente, necesita establecer e impulsar actividades de investigación a través de programas multidisciplinarios, donde representantes de las industrias e investigadores de las universidades, atiendan las diferentes problemáticas relacionadas con la producción, poniendo énfasis en la búsqueda de soluciones para lograr la automatización de las empresas, lo cual redundará en un mejoramiento de la calidad, en elevar la producción y bajar los costos.

En este contexto, el ITESCA tienen una gran responsabilidad histórica; es decir, las Universidades deben promover no solo la formación de profesionales que dominen las metodologías ingenieriles y la aplicación de las tecnologías de punta, sino también deben coadyuvar a la solución del problema de rezago tecnológico, a la modernización de la planta productiva, a través del fomento de programas de posgrado y para el desarrollo de la ciencia y tecnología, todo lo cual deberá ser reflejado en el mejoramiento de la producción y del bienestar social". [ITESCA]

Una de las acciones emprendidas por el ITESCA para coadyuvar a la innovación y al desarrollo tecnológico de México ha sido la creación del Centro de Tecnología Avanzada (CETA), donde se

generan diferentes programas para atender la problemática relacionada con el impulso del desarrollo industrial, considerando desde actividades de desarrollo tecnológico e investigación, hasta actividades de capacitación y formación de recursos de alto nivel.

Actualmente el ITESCA cuenta con equipamiento de tecnología para las áreas de electrónica, robótica y manufactura; dentro de este equipamiento se encuentra una celda de manufactura flexible integrada entre otros equipos por: un robot CRS A465 de 6 grados de libertad para una mesa de ensamble (Figura 1), un robot CRS A465 montado sobre unos ejes de movimiento transversal y longitudinal formando 8 grados de libertad para carga y descarga (Figura 2), un robot cartesiano neumático para mesa de ensamble (Figura 3).

Figura 1. Robot CRS A465 para mesa de ensamble.

Figura 2. Robot CRS A465 para carga y descarga.

Figura 3. Robot cartesiano neumático para ensamble.

Para el entrenamiento de los alumnos en el manejo y programación de robots, se hace uso de los dos robots CRS con que cuenta el ITESCA; sin embargo para realizar un entrenamiento efectivo, es necesario que los alumnos participen activamente en el manejo y la programación de los robots y

cuando los grupos de alumnos son numerosos, es un tanto imposible el permitir que todos participen elaborando personalmente la programación y el manejo del robot. Una buena medida para realizar el entrenamiento de los alumnos en forma eficiente al permitirles que cada uno elabore la programación de los robots y evaluar el funcionamiento de cada programa, sería la adquisición de programas de simulación tales que permitan a los alumnos la evaluación de sus programas al visualizar en pantalla los movimientos que realizaría el robot al ejecutar tales programas sin poner en riesgo la seguridad de las personas alrededor del robot y, sin poner en riesgo de daño físico al mismo robot. Dichos programas de simulación podrían ser instalados dentro de un laboratorio de cómputo y permitir que cada alumno tenga acceso al programa desde cualquier computadora personal.

1. ¿Qué se ha hecho en programas de simulación para robots?

Actualmente existen programas de simulación disponibles en el mundo, los cuales son de diferente complejidad, desde simples visualizadores, hasta completos sistemas de programación, simulación, comunicación, control en tiempo real de robots y sistemas integrados de varios equipos.

1.1. Simulador Encarnação

Un ejemplo de un visualizador simple es el publicado en Internet por su creador Luiz Felipe Rudge Encarnação [Encarnação], que provee de un ambiente tridimensional con gráficos en alambre con un robot móvil de 5 grados de libertad. Se puede controlar el robot por medio del ratón de la computadora al seleccionar los diferentes objetos y colocarlos en lugares válidos. Tiene posibilidad de cambiar la visualización de la escena por una de quince cámaras disponibles; incluso es posible seleccionar la vista desde una cámara montada sobre un avión que vuela en círculos alrededor del cuarto. En la Figura 4, se muestra la pantalla de trabajo de este simulador.

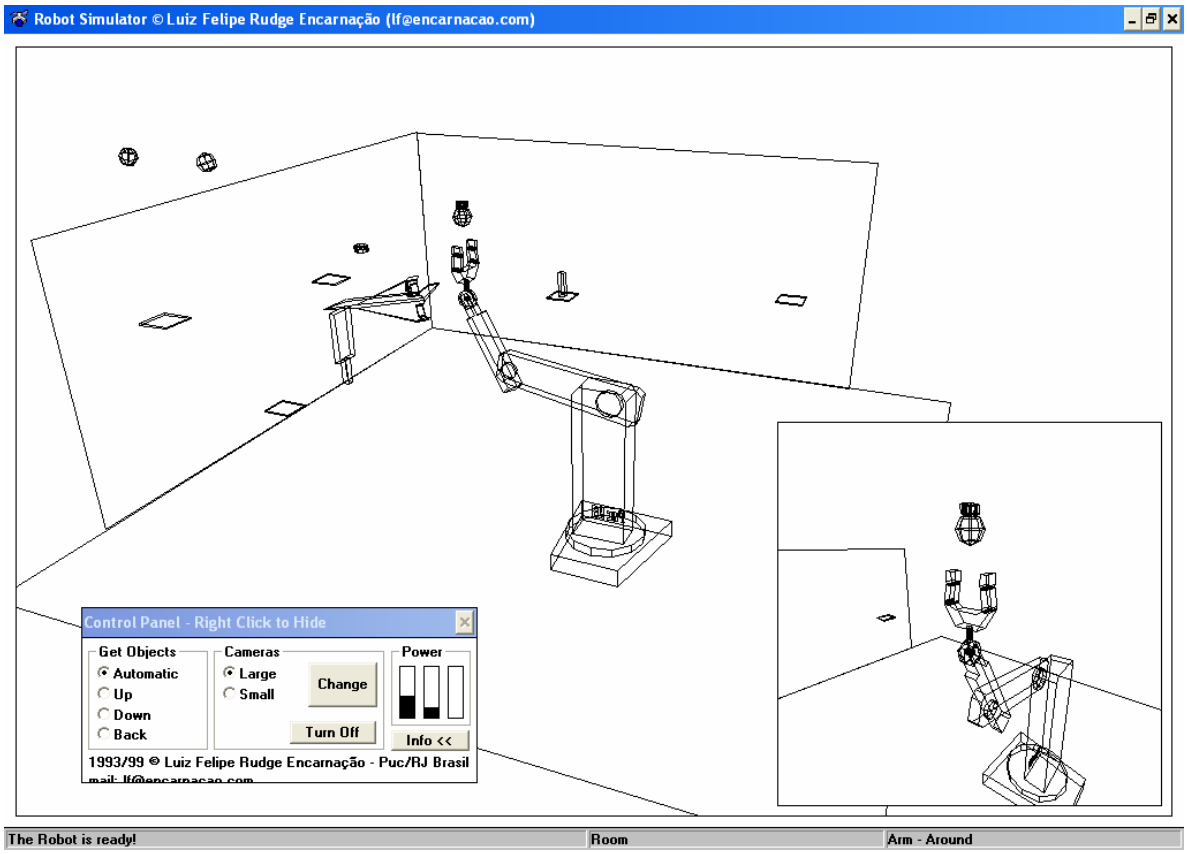


Figura 4. Pantalla del simulador Encarnação

1.2. Simulador RoboWorks

RoboWorks es un modelador tridimensional y paquete de animación para uso principalmente técnico como ambiente de simulación para sistemas mecánicos desarrollado por el Dr. Chetan Kapoor [RoboWorks]. Está diseñado para ser utilizado por ingenieros que trabajen en las áreas de control, robótica, diseño mecánico y análisis. RoboWorks es también útil como herramienta de educación. Además, RoboWorks es ideal para ser usado en automatización industrial para interfaces tridimensionales en tiempo real. RoboWorks también añade un gran valor de análisis a programas

como Matlab, LabView, Mathematica, etcétera; que tienen una excelente capacidad en graficado pero con debilidad en modelación y animación tridimensional.

RoboWorks utiliza una metodología de modelación jerárquica; simple pero poderosa. Por medio de esta metodología, el usuario interactivamente puede construir un modelo tridimensional seleccionando primitivas predefinidas. Estas primitivas pueden ser tanto cuerpos tridimensionales, transformaciones, materiales, entre otros. Por medio de esta metodología es posible crear modelos complejos, tal como el que se muestra en la Figura 5.

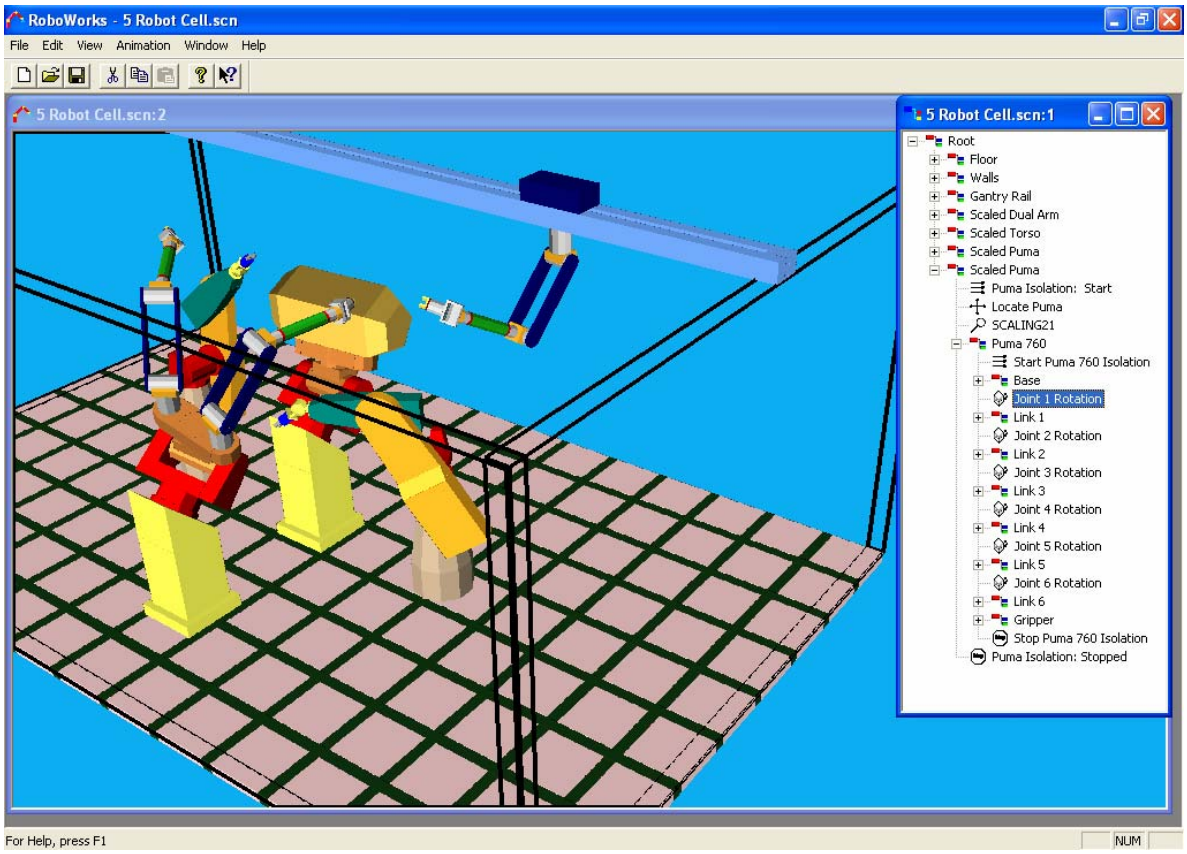


Figura 5. Pantalla del simulador RoboWorks

Hay dos formas de interactuar con RoboWorks. Estas son por medio de la pantalla tridimensional en donde se muestra el modelo en forma de sólidos sombreados; y la otra por medio de la ventana de árbol, donde se muestra la forma jerárquica del modelo. Para editar el modelo se utiliza principalmente la ventana de árbol.

La característica más potente de RoboWorks es la pantalla tridimensional interactiva en tiempo real. El modelo creado se puede animar e interactuar con él en tiempo real. Para hacer esto, RoboWorks soporta tres mecanismos diferentes:

1) Entrada por teclado: mientras se construye el modelo, el usuario puede asignar teclas como generadores de eventos. Estas teclas se pueden usar para modificar interactivamente los parámetros del modelo creando una animación. Por ejemplo, la letra “a” se puede asignar a una transformación de rotación. De manera que cuando se presione la letra “a”, la transformación de rotación cambiará su valor (por un incremento predispuesto por el usuario), lo que causará que ocurra una rotación en el modelo.

2) Archivo de datos: semejante a la entrada por teclado, el modelo se puede animar mediante un archivo de datos. Para este propósito, RoboWorks utiliza etiquetas. Estas etiquetas son nombres asignados a cada elemento de transformación en el modelo y pueden ser especificadas en un archivo de datos junto con su valor correspondiente. Se utiliza un reproductor de archivos para abrir el archivo de datos y animar el modelo.

3) RoboTalk: éste es el mecanismo de interacción en tiempo real más poderoso con que cuenta RoboWorks. RoboTalk usa el protocolo de redes TCP/IP para permitir que un programa externo se comunique con RoboWorks para controlar el modelo. RoboTalk utiliza las etiquetas que define el

usuario para las transformaciones. También existe la posibilidad de interactuar con el modelo por medio del programa LabView [National Instruments] de National Instruments.

1.3. Simulador Ropsim

Ropsim es un programa computacional para PC manejado por modelos para simulación de robots con visualización en tres dimensiones [Ropsim]. La simulación es virtual y permite la simulación de sistemas de producción en la pantalla.

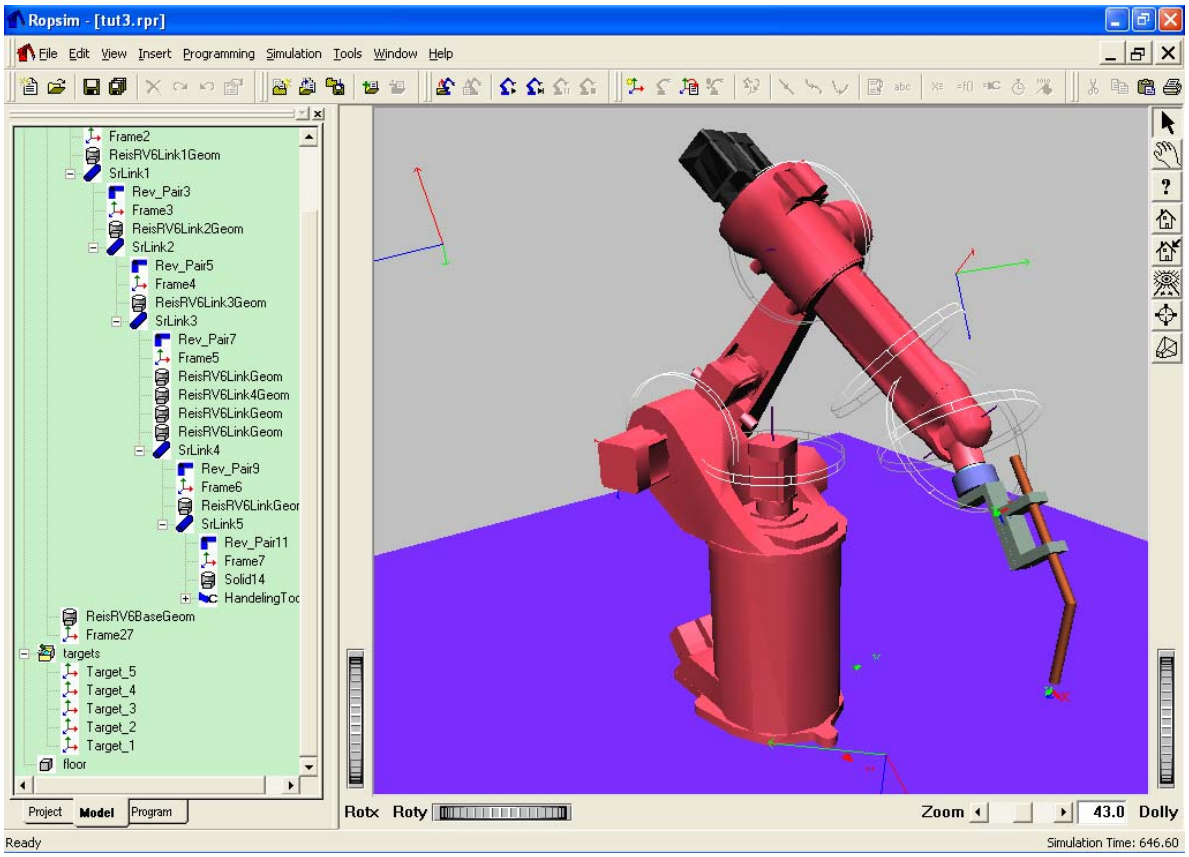


Figura 6. Pantalla del simulador Ropsim

Cuando se trabaja con planeación de producción se construye un sistema virtual tridimensional de producción en tiempo real, donde se pueden hacer experimentos y análisis sin invertir en equipo

costoso. Las simulaciones alcanzadas por Ropsim están muy cerca del 100% de similitud con la realidad.

La simulación se basa en modelos de la celda de trabajo, que incluye modelos del robot y piezas del entorno de trabajo. Estos modelos se pueden construir de diferentes formas:

1) Intercambio de modelos. Ropsim es diseñado como un componente de un sistema de producción integral, donde cada pieza de información (modelos CAD, programas de robot) es intercambiada entre los sistemas. Para ello se sigue el estándar STEP para intercambio de modelos CAD-CAM.

2) Librería de modelos. Ropsim cuenta con una librería de modelos prediseñados de robots, máquinas de control numérico, herramental, líneas de transporte, etc. Estos modelos pueden ser importados dentro de la simulación.

3) Modelos personalizados. De forma semejante a RoboWorks, Ropsim tiene una interfase gráfica para la creación de modelos sólidos virtuales. También tiene la capacidad de crear modelos de controlador para los movimientos; y ésto se hace en forma genérica parametrizada. Los programas para robots se realizan en el lenguaje de programación IRL (Industrial Robot Language, estándar DIN), que es un lenguaje basado en Pascal.

La simulación se realiza en base a los modelos y se visualiza en gráficos tridimensionales en tiempo real. El sistema de producción virtual y los modelos son evaluados en base a los resultados de la simulación. Después de obtener un buen resultado en la simulación, el programa se puede enviar electrónicamente al sistema de producción.

La programación del robot se puede hacer de dos formas: en línea y fuera de línea. La programación en línea requiere usar los equipos y robot del sistema de producción por lo que es necesario interrumpir las actividades productivas del sistema; mientras que la programación fuera de línea se basa en los modelos y no es necesario interrumpir la producción mientras se realiza y se evalúa el programa. Estas dos formas de programación se pueden combinar para optimizar la técnica de programación y a esto se le llama programación híbrida.

2. Representación plana de cuerpos tridimensionales

Los cuerpos en el espacio (llámese espacio al entorno tridimensional de nuestro mundo) pueden ser representados por elementos gráficos y pueden ser presentados en una pantalla al proyectarlos sobre un plano. Para desarrollar un programa de simulación para cuerpos tridimensionales localizados en el espacio, es necesario conocer las diferentes formas de representación de dichos cuerpos sobre planos; con el fin de que estos puedan ser proyectados sobre la pantalla. En este capítulo se hace una explicación de los diferentes tipos de proyecciones que se usan para representar objetos tridimensionales en planos.

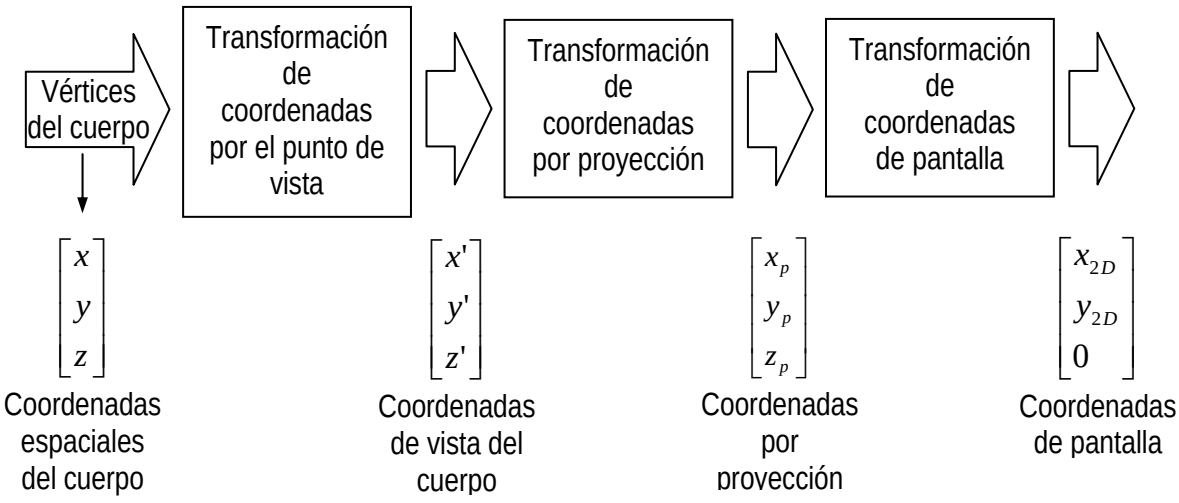


Figura 7. Conversión 3D a 2D

Existen varias formas de representación de elementos tridimensionales proyectados sobre un plano; como son la proyección paralela y la proyección perspectiva.

2.1. Proyección paralela

Define un volumen de visualización rectangular, cuyo tamaño no cambia de un extremo a otro. Se proyectan puntos de la superficie de los objetos a lo largo de líneas paralelas sobre el plano de proyección. Se especifica a partir de un vector que define la dirección de las líneas de proyección.

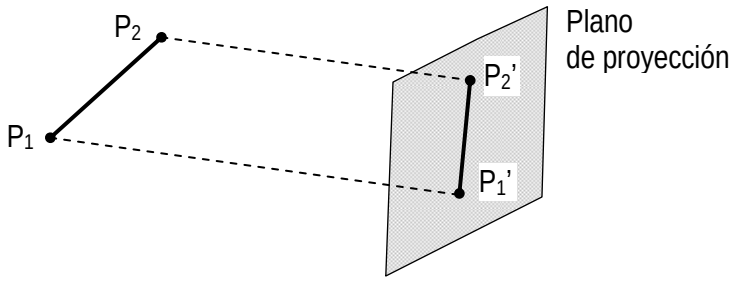


Figura 8. Proyección paralela de una línea sobre un plano

Este tipo de proyección representa dimensiones exactas de los objetos; es decir, es una representación NO realista del aspecto del objeto tridimensional. Es generalmente utilizada para representar objetos a partir de vistas que conservan las proporciones relativas; como son diseños arquitectónicos y de ingeniería. Existen varios tipos de proyecciones paralelas; a saber, proyección ortogonal y oblicua.

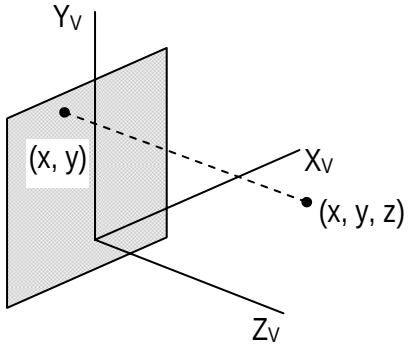


Figura 9. Proyección paralela ortogonal de un punto

1) Proyección ortogonal. Proyección perpendicular al plano de visión; teniendo las coordenadas en tres dimensiones de un punto, se puede obtener su proyección sobre el plano XY al suprimir su coordenada Z. Utilizada para generar las vistas frontal, lateral (elevaciones) y superior (vista de planta) de los objetos.

La proyección ortogonal puede a su vez ser dividida en proyección ortogonal axonométrica y proyección isométrica.

a) Proyección ortogonal axonométrica. Despliega varias caras de un objeto. Los factores de escala pueden cambiar para las 3 direcciones.

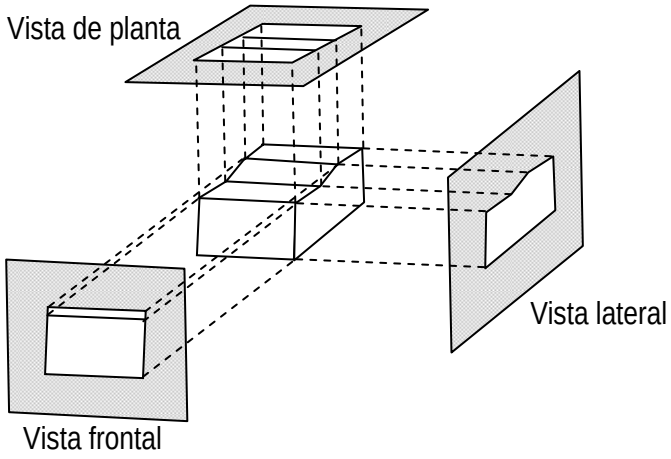


Figura 10. Proyección ortogonal axonométrica de un cuerpo

b) Proyección isométrica. Se alinea el plano de proyección de forma que cruce cada eje de coordenadas en que se define el objeto a igual distancia del origen. Se puede observar una vista del sistema de coordenadas de tal forma que se ven tres líneas rectas con un origen común y separadas a 120º. En la Figura 11 se puede observar a la izquierda un cubo ubicado en el espacio con sus vértices numerados para facilitar su interpretación; y en el lado derecho se observa la proyección isométrica del mismo cubo. Nótese la orientación de los vértices numerados.

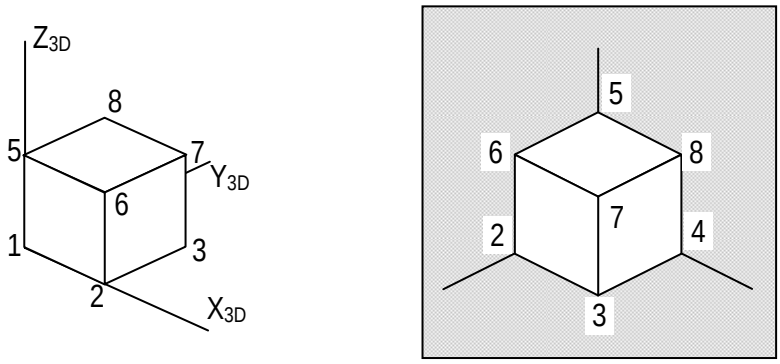


Figura 11. Proyección isométrica de un cuerpo

2) Proyección oblicua. Proyección no perpendicular al plano de visión. Este tipo de proyección puede a su vez ser dividida en proyección caballera y proyección de gabinete.

a) Proyección caballera. Todas las líneas perpendiculares al plano de proyección se proyectan sin alterar su longitud.

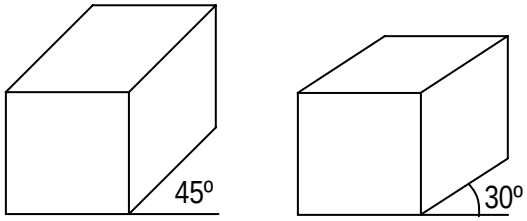


Figura 12. Proyección caballera

b) Proyección de gabinete. Las líneas perpendiculares a la superficie de vista se proyectan alterando su longitud a la mitad.

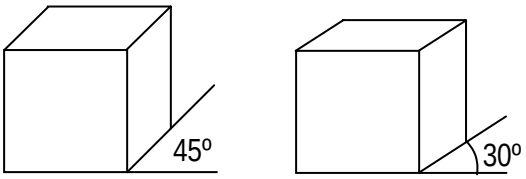


Figura 13. Proyección de gabinete

2.2. Proyección perspectiva

Representación realista del aspecto del objeto tridimensional. NO conserva las proporciones relativas de los objetos. Utilizada en animaciones y aplicaciones que requieren algún grado de realismo. En objetos del mismo tamaño, la proyección de los más próximos al plano es mayor que la de los más alejados.

Se proyectan los puntos del objeto hacia el plano de despliegue a lo largo de líneas que convergen en un punto (centro de proyección). La vista que se proyecta de un objeto se define calculando la intersección de las líneas de proyección con el plano de visión.

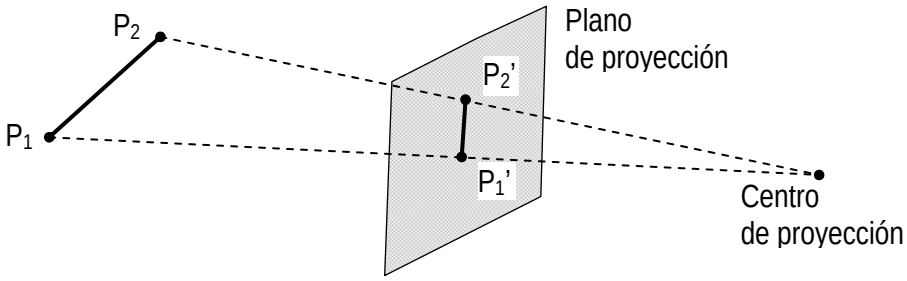


Figura 14. Proyección perspectiva de una línea sobre un plano

Para este tipo de proyección se utiliza lo que se conoce como punto de fuga; que es el punto en el que converge un conjunto de líneas paralelas que se proyectan. (Parece que convergen hacia un punto lejano en el fondo). Otro tipo de punto de fuga es el punto de fuga principal; que es en sí líneas paralelas a uno de los ejes principales de un objeto (ejes X, Y o Z). Las proyecciones en perspectiva se clasifican como de uno, dos o tres puntos de fuga principales.

3. Proyección paralela del espacio tridimensional con un punto de vista arbitrario

Una de las formas de presentación de los cuerpos tridimensionales proyectados sobre un plano; como se vio en el capítulo anterior, es la proyección paralela. Que si bien no es una representación muy realista del entorno tridimensional; como lo sería la proyección perspectiva, da una idea muy acertada del objeto a representar.

Con el fin de tener una referencia de comparación de un método de representación de objetos tridimensionales sobre una pantalla plana, en este capítulo se explica un método de desarrollo de la proyección paralela ortogonal para representar los cuerpos en el espacio tridimensional (del mundo); para después en el capítulo siguiente compararlo con la utilización de una librería OpenGL®.

Para el desarrollo del método de proyección paralela ortogonal, primero, se define un punto de vista cualquiera ubicado arbitrariamente en el espacio; lo que dará un plano de proyección X_{2D}-Y_{2D}.

Sea entonces que el plano de proyección forme un sistema de coordenadas cartesianas con su origen coincidente con el origen del sistema tridimensional original, el cual define el espacio; de manera que el vector de localización del punto de vista P_V caiga sobre el eje Z del sistema ortogonal considerado como X_{2D}-Y_{2D}-Z_{2D}. Como se muestra en la Figura 15. Nótese que al utilizar la proyección paralela y no la perspectiva, dentro del proceso de transformación 3D a 2D mostrado en la Figura 7, se está eliminando la transformación por proyección.

Se está considerando una notación 2D como subíndices tanto de los ejes X y Y como también del eje Z; considerando que cualquier punto u objeto en el espacio podrá ser representado sobre el plano de proyección X_{2D}-Y_{2D} eliminando cualquier componente o coordenada sobre el eje Z_{2D}. Nótese también de la Figura 7 que la transformación a coordenadas de pantalla se logra haciendo cero cualquier componente Z_{2D}.

En lo sucesivo, la notación a utilizar para el sistema de coordenadas cartesianas mencionado X_{2D}-Y_{2D}-Z_{2D}, se denotará como X'-Y'-Z'.

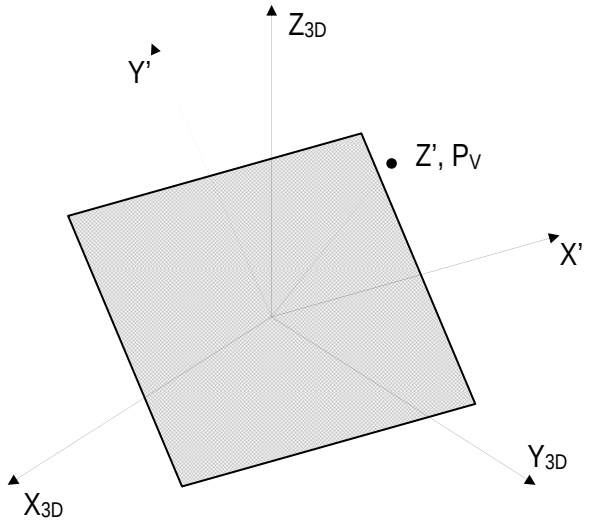


Figura 15. Proyección paralela ortogonal del espacio tridimensional con un punto de vista arbitrario

3.1. Definición del plano de proyección

Para definir el plano de proyección es necesario definir un segundo sistema de coordenadas cartesianas que representan las coordenadas en dos dimensiones visto desde un punto de vista ubicado arbitrariamente en el espacio; y para lo cual, se siguen los siguientes pasos:

Paso 1. Se define un sistema de coordenadas cartesianas secundario de forma coincidente con el sistema tridimensional original; pero de forma que el eje X' que corresponde con el eje X del sistema secundario coincida con el eje Y_{3D} que corresponde con el eje Y del sistema tridimensional original; que el eje Y' que corresponde con el eje Y del sistema secundario coincida con el eje Z_{3D} que corresponde con el eje Z del sistema tridimensional original; y por último, que el eje Z' que corresponde con el eje Z del sistema secundario coincida con el eje X_{3D} que corresponde con el eje X del sistema tridimensional original como se muestra en la Figura 16.

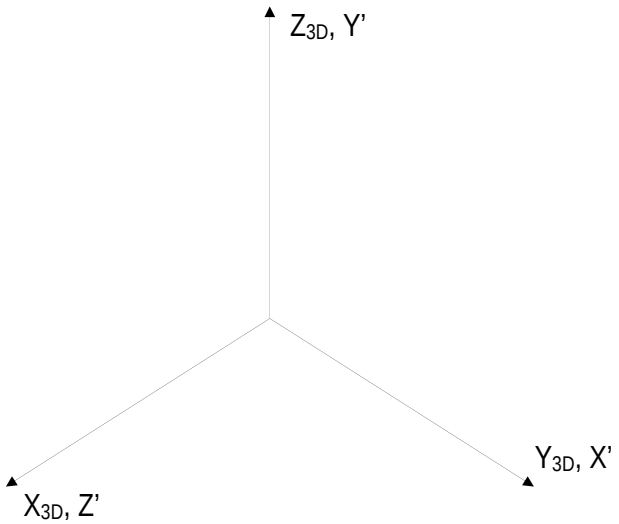


Figura 16. Creación de un segundo sistema de coordenadas (Paso 1)

Paso 2. Se hace rotar el sistema secundario un ángulo α alrededor del eje Z_{3D}; tal como se muestra en la Figura 17. Nótese que se forma un ángulo α entre el eje Z' y el eje X_{3D}; así también entre el eje X' y el eje Y_{3D}.

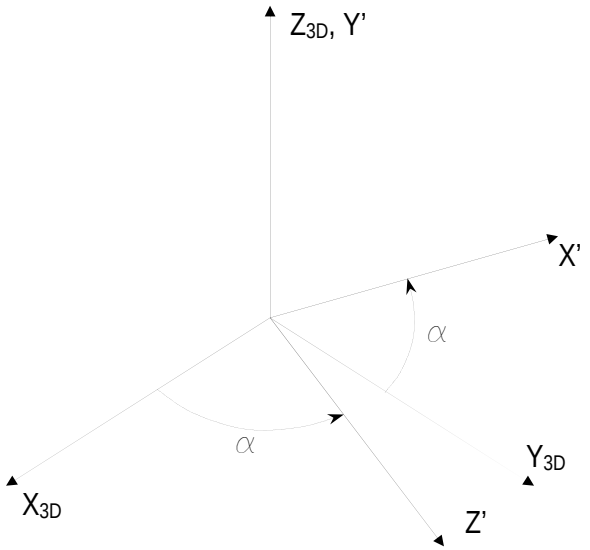


Figura 17. Creación de un segundo sistema de coordenadas (Paso 2)

Paso 3. Se hace rotar el sistema secundario un ángulo β alrededor del eje X'; tal como se muestra en la Figura 18.

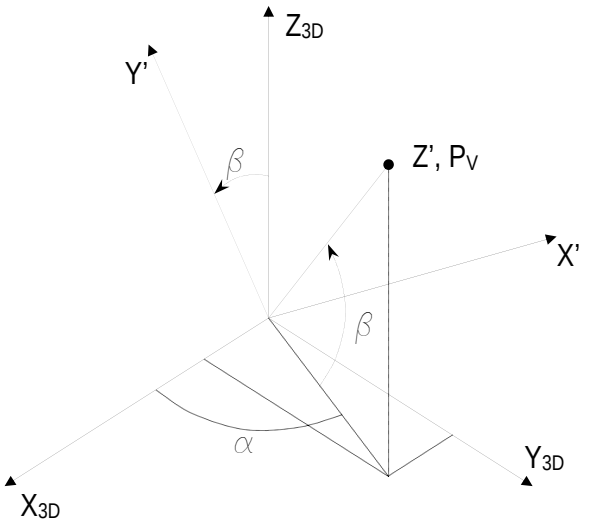


Figura 18. Creación de un segundo sistema de coordenadas (Paso 3)

Nótese que se formaría un ángulo β entre el eje Z' y el plano X_{3D}-Y_{3D}; así también, se formaría el mismo ángulo β entre el eje Y' y el eje Z_{3D}.

Comparando la Figura 18 con la Figura 15, es posible ver que el plano de proyección (que sería la pantalla) es el plano formado por los ejes X' y Y' y el punto de vista P_V se localiza sobre el eje Z'.

Habiendo creado un segundo sistema de coordenadas formando el plano de proyección definido a partir de la localización de un punto de vista arbitrario, el problema se convierte entonces en encontrar la forma de transformar la representación de los puntos que definen el objeto en el espacio para ser representados en el segundo sistema de coordenadas. Se propone entonces encontrar una matriz de transformación para realizar el cambio de base que se requiere para tal efecto. Este método se explica a continuación.

3.2. Matrices de transformación

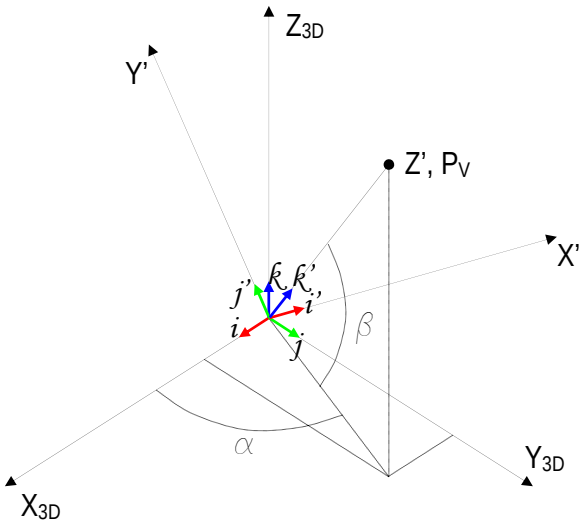


Figura 19. Declaración de vectores canónicos.

Un punto cualquiera en el espacio puede ser descrito como una combinación lineal de los vectores unitarios i, j y k del sistema tridimensional original; así como una combinación lineal de los vectores unitarios i', j' y k' del sistema secundario.

$$P = P_X i + P_Y j + P_Z k = P_{X'} i' + P_{Y'} j' + P_{Z'} k' \tag{1}$$

De la Figura 18, se puede deducir que:

$$\begin{aligned} i' &= (-\sin\alpha) i + (\cos\alpha) j \\ j' &= (-\cos\alpha \sin\beta) i + (-\sin\alpha \sin\beta) j + (\cos\beta) k \\ k' &= (\cos\alpha \cos\beta) i + (\sin\alpha \cos\beta) j + (\sin\beta) k \end{aligned} \tag{2}$$

Sustituyendo la ecuación 2 en la ecuación 1, se tiene:

$$\begin{aligned} P = P_{X'}\left((-\sin\alpha) i + (\cos\alpha) j\right) + P_{Y'}\left((-\cos\alpha \sin\beta) i + (-\sin\alpha \sin\beta) j + (\cos\beta) k\right) + \\ + P_{Z'}\left((\cos\alpha \cos\beta) i + (\sin\alpha \cos\beta) j + (\sin\beta) k\right) \end{aligned} \tag{3}$$

Reordenando se obtiene:

$$\begin{aligned} P = (-P_{X'} \sin\alpha - P_{Y'} \cos\alpha \sin\beta + P_{Z'} \cos\alpha \cos\beta) i + \\ + (P_{X'} \cos\alpha - P_{Y'} \sin\alpha \sin\beta + P_{Z'} \sin\alpha \cos\beta) j + (P_{Y'} \cos\beta + P_{Z'} \sin\beta) k \end{aligned} \tag{4}$$

Comparando la ecuación 4 con la ecuación 1 se tiene que:

$$\begin{aligned} P_X &= -P_{X'} \sin\alpha - P_{Y'} \cos\alpha \sin\beta + P_{Z'} \cos\alpha \cos\beta \\ P_Y &= P_{X'} \cos\alpha - P_{Y'} \sin\alpha \sin\beta + P_{Z'} \sin\alpha \cos\beta \\ P_Z &= P_{Y'} \cos\beta + P_{Z'} \sin\beta \end{aligned} \tag{5}$$

Representando la ecuación 5 en forma matricial, se tiene:

$$\begin{pmatrix} P_X \\ P_Y \\ P_Z \end{pmatrix} = \begin{pmatrix} -\sin\alpha & -\cos\alpha \sin\beta & \cos\alpha \cos\beta \\ \cos\alpha & -\sin\alpha \sin\beta & \sin\alpha \cos\beta \\ 0 & \cos\beta & \sin\beta \end{pmatrix} \begin{pmatrix} P_{X'} \\ P_{Y'} \\ P_{Z'} \end{pmatrix} \tag{6}$$

La ecuación 6 puede ser interpretada de la siguiente forma: si se tiene un vector representado por componentes en el sistema secundario X'-Y'-Z', este puede ser representado en el sistema tridimensional original al multiplicarlo por la matriz de transformación de la ecuación 7:

$$T_{3D} = \begin{pmatrix} -\sin\alpha & -\cos\alpha\sin\beta & \cos\alpha\cos\beta \\ \cos\alpha & -\sin\alpha\sin\beta & \sin\alpha\cos\beta \\ 0 & \cos\beta & \sin\beta \end{pmatrix} \tag{7}$$

De manera que:

$$P_{3D} = T_{3D} P_{2D} \tag{8}$$

donde P_{3D} es el vector de localización del punto P representado en el sistema tridimensional X-Y-Z, P_{2D} es el vector de localización del mismo punto representado en el sistema secundario X'-Y'-Z' o sistema de proyección en dos dimensiones. Nótese que este vector tiene componente en Z'; sin embargo, debido a que se está manejando una proyección paralela sobre el plano X'-Y' visto desde un punto de vista P_V localizado sobre el eje Z', para la graficación sobre el plano de proyección solo se utilizarían sus componentes en X' y Y'.

De igual manera se puede decir que:

$$\begin{aligned} i &= (-\sin\alpha)i' + (-\cos\alpha\sin\beta)\,j' + (\cos\alpha\cos\beta)k' \\ j &= (\cos\alpha)i' + (-\sin\alpha\sin\beta)\,j' + (\sin\alpha\cos\beta)k' \\ k &= (\cos\beta)\,j' + (\sin\beta)k' \end{aligned} \tag{9}$$

Sustituyendo la ecuación 9 en la ecuación 1, tenemos:

$$P = Px\big((-\sin\alpha)i'+(-\cos\alpha\sin\beta)\,j'+(\cos\alpha\cos\beta)k'\big)+ \\ +Py\big((\cos\alpha)i'+(-\sin\alpha\sin\beta)\,j'+(\sin\alpha\cos\beta)k'\big)+Pz\big((\cos\beta)\,j'+(\sin\beta)k'\big) \tag{10}$$

Reordenando se obtiene:

$$P = (-P_X\sin\alpha+P_Y\cos\alpha)i'+(-P_X\cos\alpha\sin\beta-P_Y\sin\alpha\sin\beta+P_Z\cos\beta)\,j'+ \\ +(P_X\cos\alpha\cos\beta+P_Y\sin\alpha\cos\beta+P_Z\sin\beta)k' \tag{11}$$

Comparando la ecuación 11 con la ecuación 1 se tiene que:

$$\begin{aligned} P_{X'} &= -P_X\sin\alpha+P_Y\cos\alpha \\ P_{Y'} &= -P_X\cos\alpha\sin\beta-P_Y\sin\alpha\sin\beta+P_Z\cos\beta \\ P_{Z'} &= P_X\cos\alpha\cos\beta+P_Y\sin\alpha\cos\beta+P_Z\sin\beta \end{aligned} \tag{12}$$

Representando la ecuación 12 en forma matricial, se tiene:

$$\begin{pmatrix} P_{X'} \\ P_{Y'} \\ P_{Z'} \end{pmatrix} = \begin{pmatrix} -\sin\alpha & \cos\alpha & 0 \\ -\cos\alpha\sin\beta & -\sin\alpha\sin\beta & \cos\beta \\ \cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta \end{pmatrix} \begin{pmatrix} P_X \\ P_Y \\ P_Z \end{pmatrix} \tag{13}$$

La ecuación 13 puede ser interpretada de la siguiente forma: si se tiene un vector representado por componentes en el sistema tridimensional original X-Y-Z, este puede ser representado en el sistema secundario X'-Y'-Z' al multiplicarlo por la matriz de transformación de la ecuación 14:

$$T_{2D} = \begin{pmatrix} -\sin\alpha & \cos\alpha & 0 \\ -\cos\alpha\sin\beta & -\sin\alpha\sin\beta & \cos\beta \\ \cos\alpha\cos\beta & \sin\alpha\cos\beta & \sin\beta \end{pmatrix} = T_{3D}^{T} \tag{14}$$

Que es igual a la transpuesta de la matriz T_{3D}.

Para el propósito de este trabajo, interesa la utilización de la ecuación 13; es decir, teniendo las componentes de puntos distribuidos en el espacio tridimensional X-Y-Z; los cuales representarían vértices de formación de cuerpos en el espacio, se puede encontrar su representación de componentes vistos desde un punto de vista cualquiera en el espacio y ser proyectados sobre un plano X'-Y' que sería la pantalla.

3.3. Cálculo de los ángulos de localización del punto de vista

Hasta ahora se han encontrado las matrices de transformación para representar las componentes de un vector de localización de un punto en el espacio, tanto para encontrar sus componentes en el sistema tridimensional del espacio a partir de sus componentes en el sistema secundario de proyección en dos dimensiones, como para encontrar sus componentes en el sistema secundario de proyección en dos dimensiones a partir de sus componentes en el sistema tridimensional del espacio. Falta entonces encontrar la forma de calcular los ángulos α y β correspondientes a un punto de vista P_V dado.

3.3.1. Producto vectorial de dos vectores

Para calcular los ángulos α y β es necesario calcular las coordenadas de los ejes X' y Y', y para ello primero se definirá el producto vectorial o producto cruz entre dos vectores.

Si se tienen dos vectores en el espacio de la forma:

$$\begin{aligned}\mathbf{A} &= A_X i + A_Y j + A_Z k \\ \mathbf{B} &= B_X i + B_Y j + B_Z k\end{aligned} \qquad (15)$$

El producto vectorial o producto cruz entre los vectores **A** y **B** se define por:

$$\mathrm{A}\times\mathrm{B}=\begin{vmatrix} i & j & k \\ A_X & A_Y & A_Z \\ B_X & B_Y & B_Z \end{vmatrix}=(A_Y B_Z - A_Z B_Y)i-(A_X B_Z - A_Z B_X)j+(A_X B_Y - A_Y B_X)k \qquad (16)$$

la ecuación 16 representa un vector perpendicular a los vectores **A** y **B** y en el sentido determinado por la regla de la mano derecha; de forma que, si se coloca el dedo índice apuntando en sentido del vector A y el dedo medio en sentido del vector B, entonces la dirección del vector AxB estará dada por el dedo pulgar extendido en forma perpendicular a los dos dedos.

3.3.2. Producto interno de dos vectores

Otro cálculo necesario para la definición de los ángulos α y β es el producto interno o producto punto entre dos vectores; también conocido como producto escalar. El producto punto entre dos vectores se define como:

$$\mathbf{A}\cdot\mathbf{B}=|\mathbf{A}||\mathbf{B}|\cos\theta \qquad (17)$$

Es decir, el producto punto entre dos vectores se define como el producto entre las normas de los vectores por el coseno del ángulo entre ellos.

Otra forma de representar el producto punto es como se muestra en la ecuación 18.

$$\mathbf{A}\cdot\mathbf{B}=(A_X i+A_Y j+A_Z k)\cdot(B_X i+B_Y j+B_Z k) \qquad (18)$$

Desarrollando el segundo término, se tiene:

$$\mathbf{A}\cdot\mathbf{B} = (A_X i\cdot B_X i + A_X i\cdot B_Y j + A_X i\cdot B_Z k + A_Y j\cdot B_X i + \\ + A_Y j\cdot B_Y j + A_Y j\cdot B_Z k + A_Z k\cdot B_X i + A_Z k\cdot B_Y j + A_Z k\cdot B_Z k) \quad (19)$$

Reordenando se tiene:

$$\mathbf{A}\cdot\mathbf{B} = A_X B_X (i\cdot i) + A_X B_Y (i\cdot j) + A_X B_Z (i\cdot k) + A_Y B_X (j\cdot i) + \\ + A_Y B_Y (j\cdot j) + A_Y B_Z (j\cdot k) + A_Z B_X (k\cdot i) + A_Z B_Y (k\cdot j) + A_Z B_Z (k\cdot k) \quad (20)$$

Tenemos que $i\cdot i = j\cdot j = k\cdot k = 1$ porque el coseno del ángulo formado es igual a 1; y además, $i\cdot j = i\cdot k = j\cdot i = j\cdot k = k\cdot i = k\cdot j = 0$ porque los vectores *i*, *j* y *k* son ortogonales. Entonces,

$$\mathbf{A}\cdot\mathbf{B} = A_X B_X + A_Y B_Y + A_Z B_Z \quad (21)$$

3.3.3. Ángulo entre dos vectores

De la ecuación 17 obtenemos que:

$$\theta = \cos^{-1}\left(\frac{\mathbf{A}\cdot\mathbf{B}}{|\mathbf{A}||\mathbf{B}|}\right) = \cos^{-1}\left(\frac{A_X B_X + A_Y B_Y + A_Z B_Z}{\sqrt{A_X^2 + A_Y^2 + A_Z^2}\sqrt{B_X^2 + B_Y^2 + B_Z^2}}\right) \quad (22)$$

Es decir, el ángulo entre dos vectores es igual al arco coseno del cociente del producto punto entre los vectores entre el producto de sus normas.

3.3.4. Cálculo del eje X'

Para calcular el eje X', es necesario tener las coordenadas del punto de localización del punto de vista P_V con respecto al sistema tridimensional del espacio. El eje X' es el producto cruz entre un

vector que define al eje Z_{3D} de la forma $Z_{3D} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$ y el vector de localización del punto de vista P_V.

$$X' = Z_{3D} \times P_V \tag{23}$$

3.3.5. Cálculo de eje Y'

Para calcular el eje Y, es necesario haber calculado previamente el eje X' y tener las coordenadas del punto de localización del punto de vista P_V con respecto al sistema tridimensional del espacio. El eje Y' es el producto cruz entre vector de localización del punto de vista P_V y el vector que define al eje X' calculado.

$$Y' = P_V \times X' \tag{24}$$

3.3.6. Cálculo del ángulo α

El ángulo α se calcula a partir de las coordenadas encontradas para definir el eje X' y el eje Y y se calcula a partir de la ecuación 25.

$$\alpha = \cos^{-1}\left(\frac{Y \cdot X'}{\|Y\|X'|}\right) \tag{25}$$

3.3.7. Cálculo del ángulo β

El ángulo β se calcula de forma semejante al ángulo α; a partir de las coordenadas encontradas para definir el eje Y' y el eje z y se calcula a partir de la ecuación 26.

$$\beta = \cos^{-1}\left(\frac{Z \cdot Y'}{|Z||Y'|}\right) \tag{26}$$

3.4. Modelo de alambre

La representación gráfica de un cuerpo tridimensional en una pantalla por medio de su modelo de alambre se logra al definir puntos de localización de los vértices del cuerpo con referencia al sistema tridimensional del espacio y transformarlos a sus coordenadas de pantalla o al plano de proyección y trazando líneas que representen las aristas del cuerpo. En la Figura 20 (a) se presenta la vista isométrica del modelo de alambre de un cubo de lado 1 con proyección paralela y en la Figura 20 (b) se presenta el mismo cubo visto desde un punto de vista localizado en las coordenadas (1, 2,3).

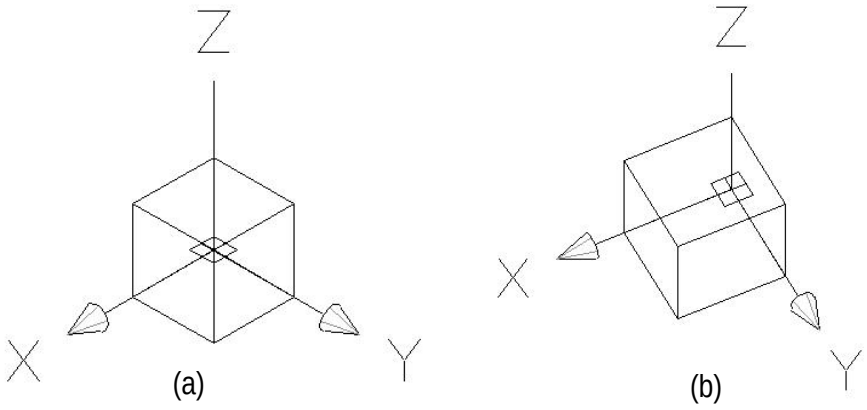


Figura 20. (a) Vista isométrica del modelo de alambre de un cubo. (b) Modelo de alambre de un cubo visto desde un punto de vista en el espacio

3.5. Sombreado

El sombreado es un método de coloreado de las caras de un cuerpo en el espacio de forma que de la apariencia de recibir cierta intensidad de luz ya sea desde una fuente de iluminación puntual o plana; de esta forma la imagen será más realista que al ser representada por su modelo de alambre o simplemente con las caras coloreadas.

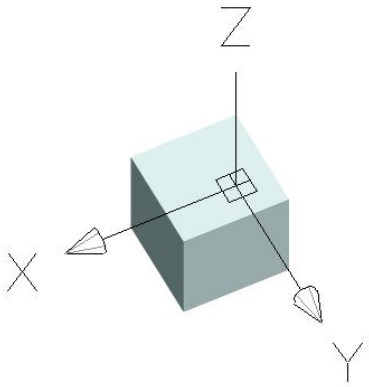


Figura 21. Modelo sombreado de un cubo visto desde un punto de vista en el espacio

Para calcular la intensidad de luz que incide en una cara del cuerpo, interesa encontrar el coseno del ángulo formado entre el vector de localización de la fuente de luz y un vector normal a la cara en cuestión.

Siendo F_l el vector de localización de la fuente de luz y N_1 un vector normal a una cara del cuerpo a sombrear; de la ecuación 17 obtenemos que:

$$\cos\theta = \frac{F_l . N_1}{|F_l||N_1|} \qquad (27)$$

La ecuación 27 dará como resultado un valor que dependiendo del ángulo formado entre los dos vectores, irá desde –1 hasta 1 y servirá como medida para saber la intensidad de la luz incidente en la cara. Es decir, para valores negativos, significa que la cara forma un ángulo mayor a 90º, o bien que se encuentra en sentido opuesto a la fuente de luz. Para un resultado con valor 0, significa que la cara forma un ángulo de 90º con la fuente de luz y por lo tanto no existe luz incidente sobre ella y por último, para valores positivos, la magnitud del resultado será una medida de la intensidad de la luz incidente sobre la cara y; por lo tanto, se puede tomar este valor y multiplicarlo por una cantidad representativa del color base de la cara dando como resultado una degradación del valor numérico del color dando una apariencia de sombreado sobre la cara.

3.6. Acomodado de caras

Cuando se pretende crear una escena con objetos con caras coloreadas, es posible que al estar dibujando las caras del objeto sobre la pantalla, si no se tiene el cuidado adecuado, se puede estar dibujando caras correspondientes a la parte posterior del objeto en la parte de encima de las caras correspondientes a la parte anterior, resultando en una imagen ambigua, como la mostrada en la Figura 22.

Para evitar lo anterior, es absolutamente necesario que cada vez que el objeto gire, o bien que el punto de vista cambie, se realice un acomodo de las caras de forma que se puedan dibujar primeramente las caras posteriores y después las anteriores. Para un objeto convexo; como por ejemplo un cubo, esto es relativamente simple. No así como para un objeto cóncavo.

Para el ejemplo del cubo, el algoritmo es el siguiente:

a) Realizar la transformación de los vértices del cubo hacia el sistema tridimensional de proyección como se muestra en la Figura 15.

b) Calcular un punto medio para cada una de las caras del cubo; el cual quedará ubicado en el cruce de las bisectrices de los cuadriláteros de cada cara.

c) Realizar una lista acomodada de menor a mayor de las caras tomando como medida de referencia la coordenada Z del punto medio calculado en el paso anterior.

d) Por último proceder a dibujar las caras coloreadas sobre la pantalla en orden creciente de menor a mayor de la coordenada Z del punto medio da cada cara.

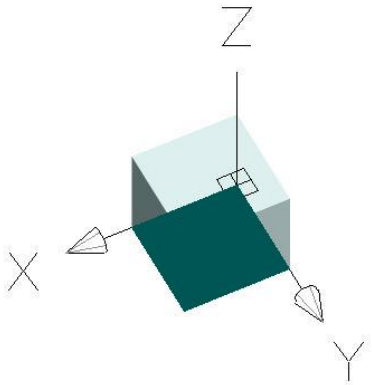


Figura 22. Imagen ambigua de un cubo sombreado

3.7. Eliminación de caras ocultas

El seguimiento fiel del algoritmo presentado anteriormente conlleva ciertamente a la creación de una imagen de un cubo sombreado; sin embargo, hasta el momento, si se observa la imagen mostrada en la Figura 21, así como si se observara el cubo desde cualquier punto de vista, se verá que en la imagen solamente se muestran tres de las seis caras del cubo; de manera que, se están dibujando

en pantalla tres de las caras que en realidad no son visibles y por lo tanto se puede decir que se invierte tiempo de procesamiento que no tiene razón de ser.

Un algoritmo apropiado al ejemplo del cubo sombreado que se ha venido comentando, sería aquel que involucra la evaluación de la coordenada Z transformada de un vector normal a cada cara del cubo; es decir si se calcula un vector normal a cada cara orientado en forma que apunte hacia fuera del cubo, se puede evaluar la componente en Z de dicho vector, de forma que si esta es positiva significa que la cara está orientada hacia el observador, por lo que se procede a dibujar y no así en el caso en que la componente es negativa; en cuyo caso significa que la cara está orientada hacia la parte de adentro de la pantalla, por lo que no se dibuja.

Para imágenes con escenas mucho más complejas; como lo son las imágenes que pudieran servir como parte de un simulador de un robot inmerso en un ambiente de trabajo, se hace entonces imprescindible, con el fin de optimizar los tiempos de procesamiento de las imágenes, el crear un algoritmo para eliminar todas aquellas caras de objetos que de cierta forma se encuentran cubiertos por otros objetos o caras de la misma escena.

4. Librería OpenGL®

4.1. Introducción

En un principio cuando se empezaron a usar gráficos en pantalla generados por computadora, el desarrollo de los programas para el manejo de estos gráficos se hacía desde cero, basados solamente en la experiencia de los programadores y no existía un estándar mundial que controlara la forma en que estos se generaban. Debido a lo anterior, cada programador utilizaba sus propios métodos y algoritmos de optimización de procesamiento; aunado a esto, era de suma importancia que las computadoras a utilizar fueran de gran capacidad de procesamiento para contrarrestar los retardos en la corrida de los programas. Actualmente con la llegada de procesadores de alta capacidad y aceleradores gráficos, es muy común ver programas de manejo de gráficos en tres dimensiones y video juegos en simples computadoras personales. Para este tiempo, los programadores se estaban enfrentando a serios problemas por la ausencia de un estándar que permitiera desarrollar programas de manejo independientes del hardware y del sistema operativo. Uno de los primeros estándares ha sido hasta hoy el conocido como OpenGL.

OpenGL es una marca registrada propiedad de la compañía Silicon Graphics Incorporated. Se ha aprobado que las especificaciones y código fuente de OpenGL estén disponibles para fabricantes y vendedores de hardware. Para usuarios finales, vendedores independientes de software y otros programadores de software basado en OpenGL no les requieren pagar por licencia de uso. [OpenGL]

OpenGL es un estándar en el área de gráficos generados por computadora y actualmente es uno de los más populares en el mundo. En 1982 en la Universidad de Stanford se desarrolló el concepto de máquina de gráficos, en donde se basó la compañía Silicon Graphics para desarrollar en su propia estación de trabajo Silicon IRIS lo que se conoce como “Rendering Pipeline” que es un orden de operaciones y plataformas de proceso. Con la base en la librería IRIS GL mencionada. En 1992 se desarrolló la librería estándar OpenGL. Los más grandes desarrolladores de hardware y software en el mundo basados en OpenGL son; entre otros, Silicon Graphics Incorporated, Microsoft Corporation, IBM Corporation, Sun Microsystems Incorporated, Digital Equipment Corporation (DEC), Evans and Sutherland, Hewlett-Packard Corporation, Intel Corporation e Intergraph Corporation.

OpenGL es una palabra compuesta por las palabras en inglés “Open Graphics Library”, que significa que es una librería para gráficos con código abierto que a pesar que fue diseñado para ser usado en los lenguajes de programación C y C++, puede ser utilizado en algunos otros lenguajes de programación, como Visual Basic, Fortran, Delphi, Pascal y Java, entre otros. Programas desarrollados con esta librería pueden ser libremente utilizados bajo sistemas operativos como Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPENStep, y BeOS; también trabaja con todos los sistemas mayores de ventanas, incluyendo Win32, MacOS, Presentation Manager, y sistema X-Window. Además es un estándar incluido de fábrica en todas las computadoras personales con Windows 95/98/2000/NT y MacOS. De esta forma libera al programador de desarrollar su aplicación para un hardware específico; es decir, si el dispositivo a utilizar soporta alguna función, entonces dicha función se ejecuta en tal dispositivo, y si no soporta la función, entonces esta se ejecuta en el procesador central. De acuerdo a Silicon Graphics Incorporated, la utilización de las rutinas de OpenGL típicamente resulta en aplicaciones con menos

líneas de código que aquellas aplicaciones utilizando otras librerías de gráficos o paquetes. [Permadi]

OpenGL es una interfase basada en procedimientos más que descriptiva; de manera que para generar una imagen coloreada de una esfera sólida, por ejemplo, el programador debe especificar la secuencia correcta de comandos para ajustar la cámara o punto de vista y las transformaciones de modelación y dibujar la esfera del color deseado. Algunos otros sistemas del tipo descriptivo, el programador simplemente especifica que se quiere dibujar una esfera de cierto color en ciertas coordenadas. La desventaja de usar una interfase del tipo basada en procedimientos es que el programa debe especificar todas las operaciones con detalle exacto y en la secuencia correcta para obtener el resultado deseado; sin embargo, la ventaja de este tipo es que permite gran flexibilidad en el proceso de generación de la imagen. El programa es libre de modificar la velocidad de procesamiento al cambiar los pasos a través de los cuales se dibuja la imagen. La forma más fácil de demostrar la potencialidad de una interfase del tipo basada en procedimientos es notar que se puede desarrollar una interfase descriptiva sobre una interfase basada en procedimientos, pero no en forma inversa.

OpenGL en concreto, es una librería de bajo nivel para gráficos que facilita el desarrollo de programas computacionales para manejo de imágenes y modelación de objetos sólidos tridimensionales ofreciendo un gran realismo a las imágenes generadas.

En el contenido del presente capítulo no se pretende proporcionar un manual para entrenamiento en la programación basada en OpenGL; sino más bien, presentar al lector una referencia informativa de las capacidades y formas de utilización de esta librería para sacar el mejor provecho de ella.

4.2. Organización típica de un programa con OpenGL

Todo programador tiene la libertad de organizar su programa de la manera que mejor le parezca; sin embargo, todo programa que utilice OpenGL para la generación de gráficos en pantalla, puede tener una organización que cuenta con algunas partes importantes y la mayoría de estas partes son indispensables dentro de la organización del programa.

En la Figura 23, se muestra un diagrama de flujo de la organización típica de un programa que use OpenGL.

4.2.1. Creación de ventana de graficado

Un programa típico que use OpenGL comienza con la apertura de una ventana dentro de una memoria intermedia de recuadro en la que se dibujará. Sin embargo, para un programa en Visual Basic no es necesario el definir un código de generación de ventana. No así como para la programación en C, por ejemplo.

4.2.2. Inicialización de contexto

OpenGL provee un juego de comandos que permiten la especificación de características que controlan cómo se presentan los objetos en la pantalla; es decir si son sombreados, la localización de la luz, el tipo de luz, el tipo de material, el tipo de proyección, la localización y orientación del observador; así como la forma como se trasladan desde el espacio tridimensional a la pantalla en dos dimensiones.

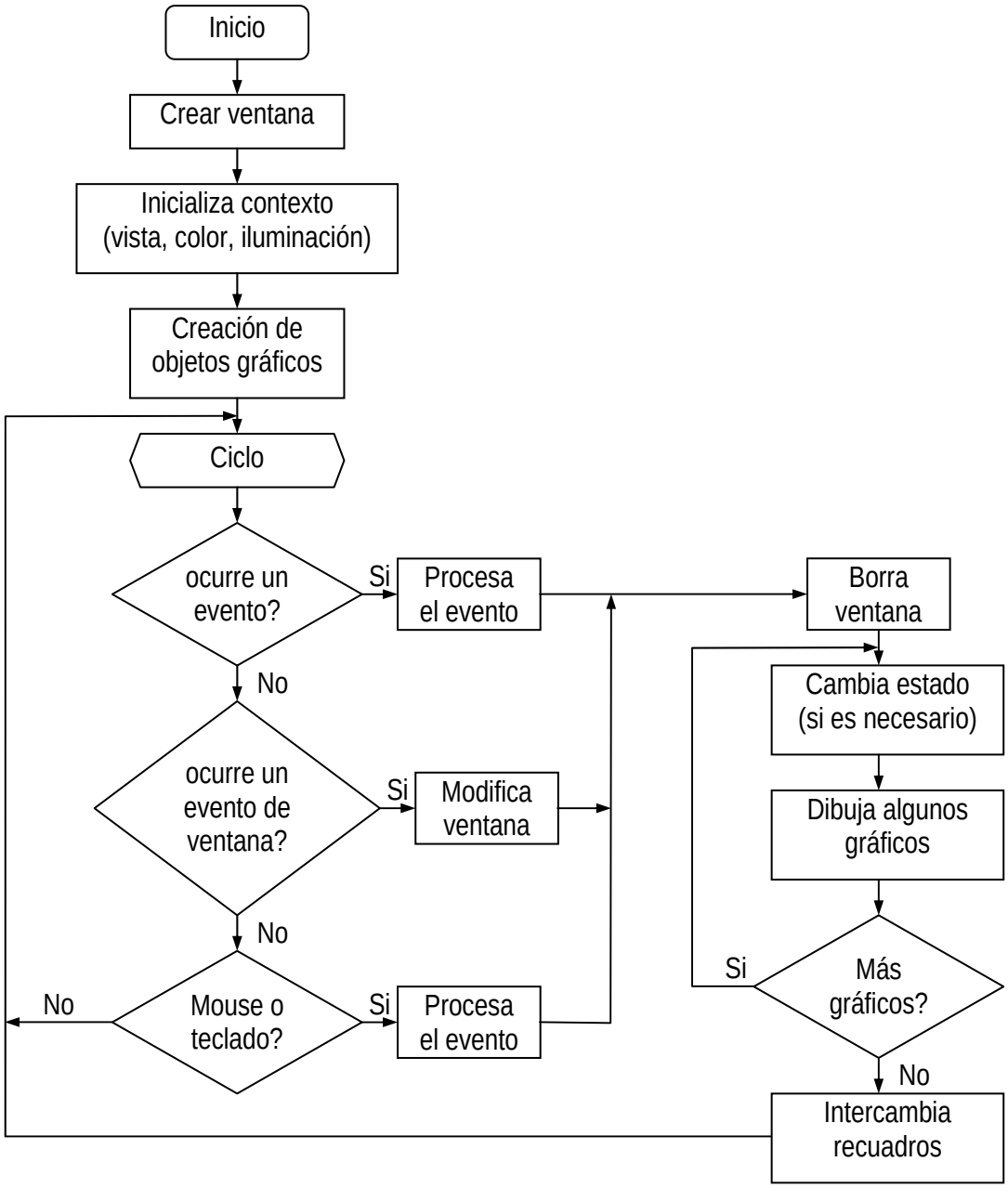


Figura 23. Diagrama de flujo de un programa típico OpenGL

Conforme se dibuja cada primitiva geométrica en la ventana, cada uno de sus vértices es afectado por el estado actual de las variables del contexto. Estas variables de estado especifican la información como el ancho de línea, el patrón de punteado de líneas, el color, el método de sombreado, la neblina, la selección de polígonos, entre otras.

Algunas variables de estado se refieren a las funciones de OpenGL que pueden estar encendidas o apagadas. Otras variables de estado se refieren a condiciones de modo; el cual se escoge de un conjunto fijo de modos disponibles. Y por último, existen algunas variables de estado que se establecen a cierto valor numérico.

Cada variable de estado tiene establecido un valor por defecto. Cada una de las variables de estado ya sea que obtengan el valor por defecto o que el programador se los asigne, permanecen activas con tal valor hasta que se cambien.

4.2.3. Creación de objetos gráficos

Una vez creado el contexto, el programador es libre de mandar comandos OpenGL para ser ejecutados en la ventana. Algunas llamadas se usan para dibujar objetos de simple geometría; como puntos, líneas o polígonos, mientras que otras; como se vio en la sección anterior, afectan la forma en que los objetos se presentan en la ventana. OpenGL pone a disposición del programador un conjunto de primitivas geométricas; como son, puntos, líneas, polígonos, imágenes y mapas de bits, tanto en dos como en tres dimensiones; con los que el programador puede crear desde cuerpos virtuales hasta escenas del ambiente o del entorno.

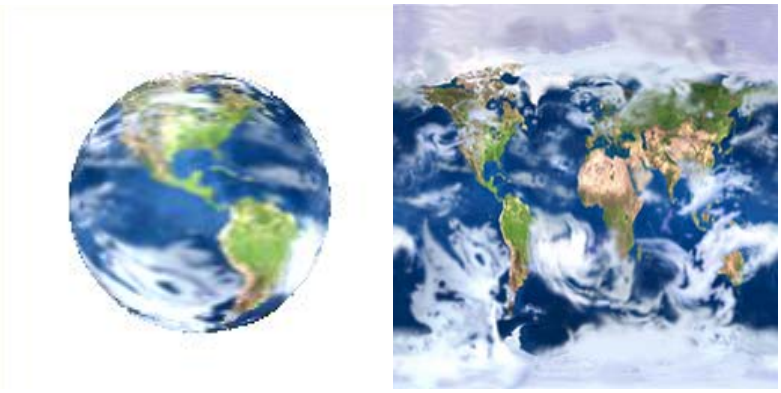
El programador tiene la opción de construir objetos geométricos utilizando cualesquiera de las siguientes primitivas:

- ✓ Puntos
- ✓ Líneas
- ✓ Secuencias de líneas
- ✓ Lazos de líneas
- ✓ Polígonos
- ✓ Cuadriláteros
- ✓ Secuencias de cuadriláteros
- ✓ Triángulos
- ✓ Secuencias de triángulos
- ✓ Abanicos de triángulos

Cada objeto geométrico se define por un conjunto de vértices y el tipo de primitiva a generar. El si se conectan y cómo se conectan los vértices se determina por el tipo de primitiva a generar.

OpenGL también soporta la visualización de imágenes en dos dimensiones como fotografías o mapas de bits, por ejemplo, y ser tratadas como cualquier otro objeto geométrico tridimensional. En

la Figura 24 se muestra un ejemplo de esto. La Figura 24 (a) presenta una imagen de una esfera forrada de una textura definida por la imagen plana de la Figura 24 (b)



(a) (b)

Figura 14. Ejemplo de utilización de una imagen 2D en un cuerpo 3D

4.2.4. El ciclo de eventos

Los programas basados en OpenGL generalmente corren en un ciclo de eventos. Esto significa que después de hacer las inicializaciones requeridas, el programa cae en un ciclo infinito aceptando el manejo de eventos. Tales eventos incluyen aquellos como presionar una tecla, movimiento del ratón apuntador, presionar un botón del ratón apuntador, soltar un botón previamente presionado del ratón apuntador, cambiar el tamaño de la ventana de dibujo, entre otros.

4.2.5. Intercambio de recuadros

La memoria de recuadro o “frame buffer” es una localidad de memoria en la computadora en donde se almacenan las imágenes al ser dibujadas; es decir es una memoria de localización de pixeles donde se establecen las intensidades y colores de cada pixel resultado de la imagen a dibujar y

proyectar en pantalla. La particularidad de OpenGL con esta memoria es que permite manejar una memoria doble de recuadro, lo cual ofrece la facilidad de que mientras que una de las imágenes es proyectada en pantalla la otra localidad de memoria puede estar en proceso de redibujado; una vez terminado el proceso de redibujado, se pueden intercambiar los recuadros para proyectar la imagen recientemente dibujada.

Esta característica permite entonces al programador crear animaciones de movimiento de objetos en pantalla al proyectar cuadros con secuencia de movimientos; de forma semejante a como se hace en cine con 24 cuadros por segundo o en televisión con 60 cuadros por segundo, en donde se aprovecha la incapacidad del ojo humano de captar los cambios repentinos de cuadro a cuadro creando la sensación de movimiento suave.

4.3. Orden de Operación OpenGL

Las etapas seguidas para la creación de escenas o imágenes en un programa típico basado en OpenGL son como sigue:

a) Construir figuras u objetos por medio de las primitivas disponibles descritas en la sección 4.2.3 Creación de objetos gráficos.

b) Acomodar los objetos creados en un espacio tridimensional.

c) Seleccionar un punto de vista en el espacio para el entorno tridimensional creado.

d) Calcular el color de los objetos. El color puede ser explícitamente asignado por el programa, puede ser determinado por las condiciones del material y la iluminación del ambiente o bien por la asignación de una textura o imagen sobre el objeto.

e) Convertir la descripción matemática de los objetos y la información de color asignada a pixeles sobre la pantalla.

Durante estas etapas, OpenGL pudiera estar realizando otras operaciones como eliminando las partes de los objetos que se encuentren escondidas por otros objetos en la escena. Como se explica en la siguiente sección.

4.4. Dibujo en 3 dimensiones

En una escena tridimensional los objetos más cercanos al observador pudieran cubrir aquellos objetos más alejados. Para dibujar una escena realista se sigue un algoritmo de eliminación de caras ocultas. A diferencia del método establecido en la sección 3.7 Eliminación de caras ocultas, OpenGL ofrece un método simple de obtener este resultado; al que se le conoce como “depth buffer” o memoria de profundidad.

Cuando se dibuja una escena, se asocia un valor de profundidad o valor Z a cada pixel y se almacena en la memoria de profundidad. Se realiza una prueba de profundidad a cada vértice en la escena; de manera que cada vez que se quiere agregar un vértice a la escena se verifica si existe algún otro vértice que cubra la vista del vértice que se quiere dibujar, en cuyo caso no se dibuja.

Por defecto OpenGL dibuja los polígonos con color de relleno sólido, tomando en cuenta los pixeles dentro del área delimitada por los lados del polígono; pero también se pueden establecer para ser

dibujados sin relleno, dibujándose solamente las líneas de los lados (modelo de alambre), o bien, en modo de puntos, dibujándose solamente los puntos que definen los vértices. Otra particularidad acerca de los polígonos es que estos tienen definido una cara frontal y una posterior. Cada lado puede ser dibujado en el mismo modo o en modos diferentes para permitir la generación de vistas de cortes de objetos sólidos por ejemplo.

4.5. Transformaciones de vista y modelo

La transformación de la vista es semejante a mover la posición y la orientación del observador; mientras que la transformación del modelo es semejante a mover la posición y orientación del modelo o de los objetos dibujados. Estas transformaciones se logran por la modificación de matrices. En cualquier momento, dentro de un programa con OpenGL se puede modificar una de las matrices de transformación; que son, la matriz del punto de vista, la matriz de proyección y la matriz de textura.

Se pueden transferir valores hacia la matriz activa para ser modificada y estos valores pueden definir transformaciones de rotación, traslación o escalamiento.

El comando glTranslate(x, y, z) multiplica la matriz activa por una matriz que mueve un objeto una distancia especificada por los componentes x, y, z dados (o mueve el origen del sistema local de coordenadas en la misma cantidad).

El comando glRotate (ángulo, x, y, z) multiplica la matriz activa por una matriz que rota un objeto (o el sistema local de coordenadas) en el sentido horario sobre el eje definido desde el origen hasta el punto de coordenadas (x, y, z), y rota un ángulo proporcionado en grados en los parámetros.

El comando glScale(x, y, z) multiplica la matriz activa por una matriz que estira, encoge o refleja un objeto a lo largo de los ejes. Cada coordenada x, y, z de cada punto en el objeto se multiplica por el argumento correspondiente x, y, o z. En caso de ser el sistema local de coordenadas el afectado, los ejes del sistema local de coordenadas se encogen con los factores correspondientes x, y, o z dados y los objetos asociados a dicho sistema local se encogen junto con el sistema.

4.6. Transformación de pantalla

Se puede hacer una transformación también que provoque un cambio del tamaño de la ventana de graficación. OpenGL por defecto establece la ventana de graficación del tamaño que es declarada al inicio del programa; no obstante, en cualquier momento durante la corrida del programa se puede reasignar el tamaño de la ventana y también se puede dejar activa la opción de que el usuario cambie el tamaño de la ventana con el uso del ratón apuntador, lo que generaría un evento de ventana del que se hace referencia en la Figura 23.

4.7. Proyección perspectiva

En proyección perspectiva, entre más alejado del observador esté un objeto, este se apreciará más pequeño. En OpenGL se consigue este efecto al utilizar un volumen de visualización en forma de una sección piramidal como se muestra en la Figura 25. Un ejemplo de proyección en perspectiva se muestra en la Figura 26 (a).

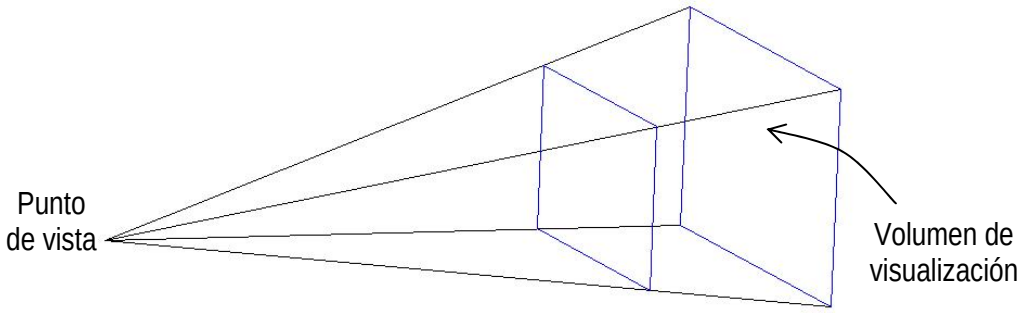


Figura 25. Volumen de visualización para proyección perspectiva

4.8. Proyección ortográfica

Con la proyección ortográfica o paralela como se le llamó en la sección 2.1 Proyección paralela, el volumen de visualización tiene forma de caja, a diferencia de la proyección perspectiva de la sección anterior, el tamaño de los objetos representados en el plano de proyección no cambian su tamaño. Por lo anterior, la distancia de posición del punto de vista no afecta el tamaño de apreciación de los objetos en la pantalla. Un ejemplo de proyección paralela se muestra en la Figura 26 (b).

Para ambos tipos de proyecciones; tanto para la proyección perspectiva como para la proyección ortográfica, se especifican los planos de definición del volumen de visualización y a estos planos se les conoce como planos de recorte. Y como su nombre lo dice, estos planos sirven para recortar el volumen de visualización, de forma que todos aquellos objetos o incluso partes de objetos que queden fuera del volumen de visualización serán recortados de la imagen proyectada en la pantalla.

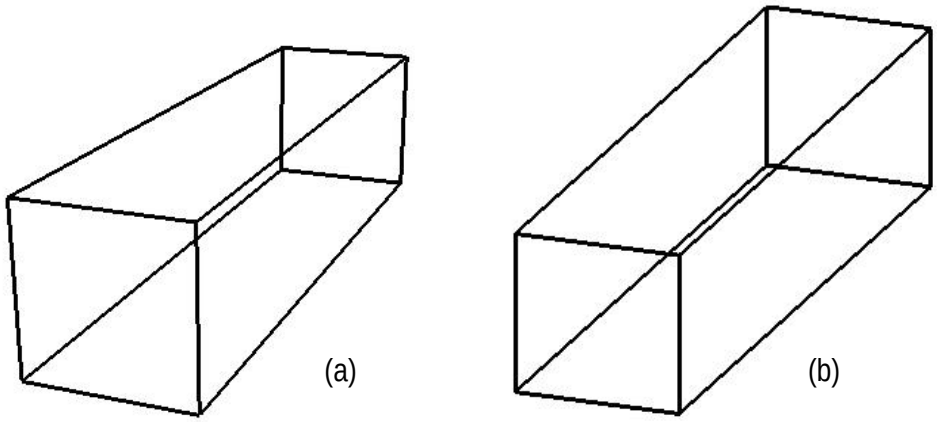


Figura 26. (a) Ejemplo de proyección perspectiva. (b) Ejemplo de proyección paralela

4.9. Animación

La animación o imágenes con objetos con apariencia de movimiento se logran a través de la utilización de la memoria doble de recuadro explicada en la sección 4.2.5 Intercambio de recuadros.

4.10. Iluminación

OpenGL ofrece la utilización de cualquiera de dos tipos de iluminación; a saber, luz direccional y posicional. Una luz direccional es considerada como una luz que proviene de una fuente de luz a una distancia infinitamente alejada del lugar de la escena. Este tipo de luz se le puede considerar como que proviene de una fuente de luz plana perpendicular al lugar de la escena; de manera que los rayos de luz se pueden considerar como paralelos cuando estos llegan a los objetos. Una luz posicional o puntual es aquella que proviene de una fuente de luz puntual o esférica cercana al lugar de la escena; de manera que los rayos de luz provienen desde el centro de la fuente de luz y se esparcen uniformemente en todas direcciones.

Para que se lleven a cabo los cálculos necesarios para lograr el efecto de iluminación, se deben especificar vectores normales; así como también las propiedades del material de cada elemento de superficie de los objetos involucrados en la escena.

En la Figura 27 se presenta un ejemplo de un objeto con efecto de iluminación puntual.

Figura 27. Balón de fútbol con efecto de iluminación puntual

4.11. Memoria de plantilla

Antes de hablar de la geometría constructiva de sólidos, explicada en la sección 4.12 Geometría constructiva de sólidos, es necesario hablar de la memoria de plantilla; que es parecida a la memoria de profundidad explicada en la sección 4.4 Dibujo en 3 dimensiones o la de color o de recuadro, como se le llamó en la sección 4.2.5 Intercambio de recuadros, con la diferencia que los bits alojados en esta memoria no representan color o profundidad y tampoco son directamente visibles; sin embargo, afectan la imagen visible de la pantalla y se actualizan a través de una función de plantilla y una operación de plantilla. La función de plantilla controla si un fragmento de la imagen es

eliminado o no por la prueba de plantilla y la operación de plantilla determina la forma en como se actualiza la memoria de plantilla como resultado de la prueba de plantilla.

La prueba de plantilla ocurre inmediatamente después de la prueba de orientación para el color e inmediatamente antes de la prueba de profundidad. Si esta prueba está habilitada, se puede controlar lo que pasa bajo tres diferentes casos:

- ✓ Si la prueba de plantilla falla
- ✓ Si la prueba de plantilla pasa, pero la prueba de profundidad falla
- ✓ Ambas pruebas, la de plantilla y profundidad pasan

Ya sea que la prueba de plantilla para un fragmento gráfico específico pase o falle no tiene efecto alguno sobre el color o el valor de profundidad del mismo fragmento. La operación de plantilla es una comparación entre el valor en la memoria de plantilla para los pixeles de destino de los fragmentos gráficos y un valor de referencia de la plantilla; es decir, se realiza una operación binaria del tipo AND con el valor en la memoria de plantilla y el valor de referencia de la plantilla antes de aplicar la comparación. Los tipos de comparaciones de plantilla se enlistan a continuación:

- ✓ Siempre falla
- ✓ Siempre pasa
- ✓ Pasa si el valor de referencia es menor al valor de la memoria de plantilla
- ✓ Pasa si el valor de referencia es menor o igual al valor de la memoria de plantilla

- ✓ Pasa si el valor de referencia es igual al valor de la memoria de plantilla
- ✓ Pasa si el valor de referencia es mayor o igual al valor de la memoria de plantilla
- ✓ Pasa si el valor de referencia es mayor al valor de la memoria de plantilla
- ✓ Pasa si el valor de referencia es diferente al valor de la memoria de plantilla

A fin de cuentas, si la prueba de plantilla falla para cierto fragmento gráfico, entonces tal fragmento se descarta y no se transfiere a la memoria de recuadro para ser proyectada a la pantalla. Si la prueba de plantilla pasa, entonces se aplica la prueba de profundidad al fragmento en cuestión; en caso de estar activada, y si esta prueba pasa, o si está desactivada, entonces se transfiere la información a la memoria de recuadro para ser proyectada en la pantalla y se aplica una operación al valor de plantilla para el pixel en prueba. Si la prueba de profundidad falla, entonces el fragmento gráfico se descarta por efecto de esta prueba; sin embargo la operación de plantilla sí se aplica al valor de la memoria de plantilla. Las operaciones de plantilla disponibles se enlistan a continuación:

- ✓ El valor de la plantilla se mantiene
- ✓ El valor de la plantilla se pone a cero
- ✓ El valor de la plantilla es reemplazado por el valor de referencia
- ✓ El valor de la plantilla se incrementa
- ✓ El valor de la plantilla se decrementa
- ✓ El valor de la plantilla se invierte

4.11.1. Disolución de imágenes

La memoria de plantilla puede ser usada para enmascarar algunos pixeles en la pantalla y esto permite crear imágenes compuestas pixel a pixel. Se puede dibujar alguna geometría o arreglos de valores en la memoria de plantilla para controlar cuales pixeles dibujar en la memoria de recuadro. Una forma de usar esta capacidad es creando imágenes compuestas.

Una técnica común en la edición de videos es disolver imágenes; de forma que una imagen o bien una secuencia animada de imágenes se reemplaza por otra en forma suave. La memoria de plantilla se puede usar para implementar patrones arbitrarios de disolución de imágenes. Una aproximación buena de esta técnica es el combinar dos imágenes diferentes, usando la memoria de plantilla para controlar cuales pixeles provenientes de ambas imágenes se dibujan en la memoria de recuadro. Esto se hace definiendo una prueba de plantilla y asociando un valor de referencia diferente para cada imagen. La memoria de plantilla se inicializa a un valor tal que la prueba de plantilla pase con el valor de referencia de una de las imágenes y falle con la otra.

Al inicio del proceso, la memoria de plantilla se limpia a un solo valor, permitiendo que una de las imágenes se dibuje en la memoria de recuadro. Cuadro a cuadro, la memoria de plantilla se va cambiando progresivamente conforme el programador lo defina a un valor diferente, de manera que permita pasar secciones de la segunda imagen para ser dibujada encima de la anterior. Ésto da como resultado que paulatinamente, la primera imagen es reemplazada por la segunda.

4.11.2. Efecto de calcomanía

En el caso de disolución de imágenes visto en la sección anterior, la memoria de plantilla controla en dónde se dibujan los pixeles provenientes de una escena completa. En esta sección se utiliza la memoria de plantilla para dibujar pixeles de una primitiva específica sobre una escena completa.

El efecto de calcomanía es aquel en que a una imagen de una escena de pantalla completa se superpone una segunda imagen con cierta figura geométrica; dando el efecto de haber pegado una calcomanía encima de la imagen anterior. La forma de hacerlo es similar al efecto de disolución de imágenes sólo que la máscara utilizada deberá dar como resultado el recorte de la imagen superpuesta de la forma de la figura geométrica deseada.

4.11.3. Imágenes compuestas con profundidad

Una buena técnica para aumentar la complejidad de una escena es crear la escena con imágenes compuestas. Para crear una imagen compuesta tridimensional, es necesario usar los valores de color y profundidad simultáneamente; de manera que la prueba de profundidad determine cuáles segmentos son cubiertos por otros más cercanos al observador. Para realizar la composición de las imágenes, primero se deshabilita la memoria de color para que no pueda dibujar, se limpia la memoria de plantilla y se copian los valores de profundidad en la memoria de recuadro. Se habilita la prueba de profundidad asegurando que sólo aquellos segmentos de imagen que se encuentren más cercanos al observador que la imagen original puedan actualizar la memoria de profundidad. La memoria de plantilla contiene ahora una máscara de los pixeles que estaban más cerca que la imagen original. Se cambia la función de la plantilla y se habilita la memoria de color para que pueda

ser dibujada. Al dibujar la memoria de color, se dibujarán solamente aquellos pixeles que cumplan con la prueba de la plantilla.

Este proceso se puede repetir cuantas veces sea necesario para añadir más imágenes a la misma escena, e incluso es posible manipular la profundidad de las imágenes por agregar.

4.12. Geometría constructiva de sólidos

La geometría constructiva de sólidos (CSG por sus siglas en inglés) es una técnica que se utiliza para representar objetos sólidos virtuales por medio de intersecciones, uniones y substracciones de objetos sólidos; estos pueden ser tanto primitivas sólidas o bien objetos creados por medio de esta misma técnica CSG. Al plasmar los operandos (objetos sólidos) y sus operaciones binarias (intersecciones, uniones y substracciones) se forma un árbol llamado árbol CSG. La simbología para representar las operaciones binarias es como sigue:

$\cap$ Para intersección

$\cup$ Para unión

$-$ Para substracción

Y las operaciones binarias con los sólidos pueden ser también representadas por una ecuación booleana. Por ejemplo la operación $(A \cup B) - C$ da como resultado el sólido de la derecha de la Figura 28, considerando que los sólidos A, B y C son los que se presentan en el árbol CSG de la misma figura.

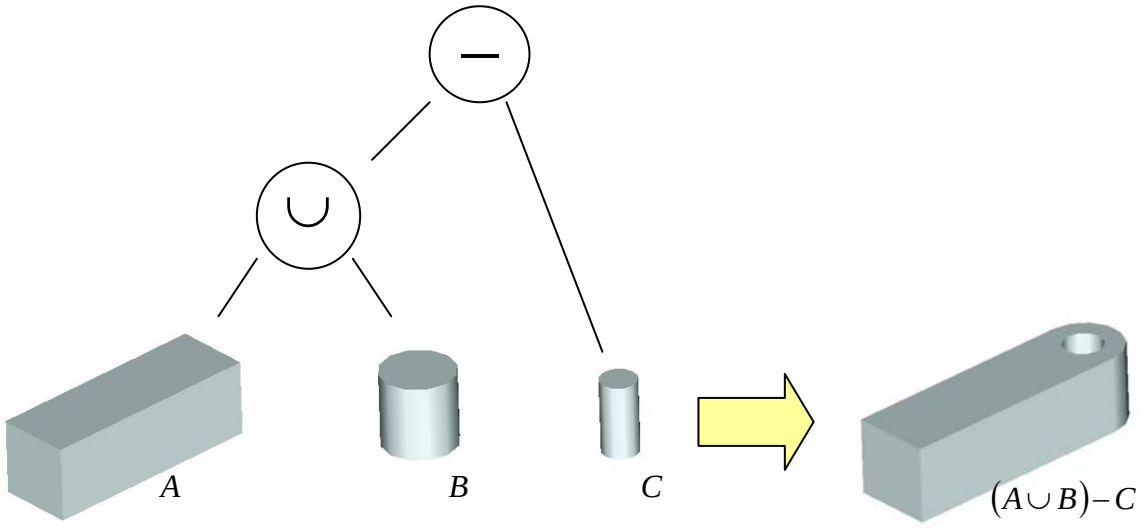


Figura 28. Ejemplo de un sólido con geometría constructiva

Desde el inicio de la era del manejo de gráficos por computadora utilizando la técnica CSG, se han propuesto algunos algoritmos para la puesta en escena de los arreglos representados por árboles CSG. Estos algoritmos para representación en pantalla de cuerpos sólidos descritos por medio de árboles CSG se pueden dividir en algoritmos basados en el espacio de la imagen y aquellos basados en el espacio de los objetos.

- ✓ Algoritmos basados en el espacio de la imagen son aquellos que utilizando la memoria de profundidad explicada en la sección 4.4 Dibujo en 3 dimensiones, generan solamente la imagen en pantalla de una figura CSG sin calcular una descripción de la geometría final del objeto [Kirsch, 2004]; de manera que todas las operaciones de unión, intersección y sustracción de las primitivas del cuerpo se hacen en el proceso de conversión del objeto a la imagen en pantalla. Debido a que estos algoritmos regeneran la solución en cada cuadro,

son mejor aplicados a situaciones donde la localización, forma o la interacción de los objetos cambian constantemente [Stewart, 1998].

- ✓ Algoritmos basados en el espacio de los objetos, son independientes del punto de vista y se dice que estos son más costosos que los basados en el espacio de la imagen [Stewart, 1998]. Las operaciones booleanas de construcción de sólidos se hacen directamente alterando las formas de las primitivas. Tienen la ventaja de que el proceso de evaluación del conjunto de objetos no tiene que ser regenerado para cada cuadro.

Los trabajos previos de representación de sólidos CSG en pantalla se pueden clasificar a su vez de acuerdo a los métodos que usan.

El método más obvio es convertir la representación CSG en representación de fronteras; sin embargo este método requiere de mucho procesamiento y es muy lento para aplicaciones donde se requiere interactividad con el modelo.

Algunos algoritmos muy conocidos se han modificado para su uso en la proyección en pantalla de modelos CSG. Incluyendo los métodos scanline, ray tracing, ray casting, image subdivision, octrees, point sampling, voxel reconstruction y el método Binary Space Partitioning (BSP).

Se han sugerido algunos otros métodos también para el mismo propósito usando hardware de propósito especial y han ido desde la utilización de memorias de profundidad especiales hasta la utilización de memorias múltiples tanto de profundidad como de recuadro. El problema de estos métodos es precisamente la utilización de hardware especial que no es nada barato ni práctico de conseguir.

Dentro de los algoritmos propuestos destacan algunos que por su aportación han sido ampliamente aceptados; como son los descritos a continuación.

4.12.1. Algoritmo Goldfeather

En 1986 Jack Goldfeather, Jeff P. M. Hulquist y Henry Fuchs presentaron un algoritmo para la puesta en escena de árboles CSG de objetos convexos [Goldfeather, 1986] y posteriormente en 1989, Jack Goldfeather, S. Molnar, G. Turk, y Henry Fuchs para objetos no convexos [Goldfeather, 1989]. En el artículo “Fast Constructive Solid Geometry Display, in the Pixel-Powers Graphics System” [Goldfeather, 1986], se muestra además un método de normalización para transformar un árbol CSG genérico en otro árbol equivalente que es una unión de árboles más simples. Se dice que un árbol CSG está normalizado cuando se encuentra en la forma de suma de productos; es decir, cuando su expresión puede ser escrita como una unión de intersecciones o sustracciones. El normalizar un árbol CSG permite que se utilice un algoritmo de proyección más simple del que sería de otra forma. En el artículo “Near Real-time CSG Rendering using Tree Normalization and Geometric Pruning”, se presenta además que teniendo un árbol CSG normalizado, se pueden identificar ramales innecesarios que se pueden “podar” para mejorar el desempeño del proceso de graficación.

El algoritmo de normalización para árboles CSG puede ser descrito con las siguientes ecuaciones de equivalencia:

$$X - (Y \cup Z) = (X - Y) - Z \tag{28}$$

$$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z) \tag{29}$$

$$X-(Y\cap Z)=(X-Y)\cup(X-Z) \tag{30}$$

$$X\cap(Y\cap Z)=(X\cap Y)\cap Z \tag{31}$$

$$X-(Y-Z)=(X-Y)\cup(X\cap Z) \tag{32}$$

$$X\cap(Y-Z)=(X\cap Y)-Z \tag{33}$$

$$(X\cup Y)-Z=(X-Z)\cup(Y-Z) \tag{34}$$

$$(X\cup Y)\cap Z=(X\cap Z)\cup(Y\cap Z) \tag{35}$$

En este algoritmo, después de normalizar el árbol CSG, cada primitiva se recorta en el espacio de la imagen por las otras primitivas usando la prueba de profundidad y de plantilla. Las superficies resultantes finalmente se mandan a la pantalla por una prueba de profundidad (z-menor-que). Esta técnica utiliza una memoria de profundidad (memoria de superficie) para almacenar la superficie visible de cada primitiva y otra (memoria de salida) para acomodar en orden correcto de profundidad los resultados parciales de la memoria de superficie. Se requiere también una memoria de plantilla para el recorte de las primitivas en la memoria de superficie. Cada vez que se compara una primitiva con la memoria de superficie, se configura la prueba de plantilla de tal forma que la memoria de plantilla retenga las superficies que se encuentren enfrente de lo almacenado en la memoria de profundidad [Theoharis, 2001].

En la Figura 29, se presenta un diagrama de flujo del algoritmo de Goldfeather.

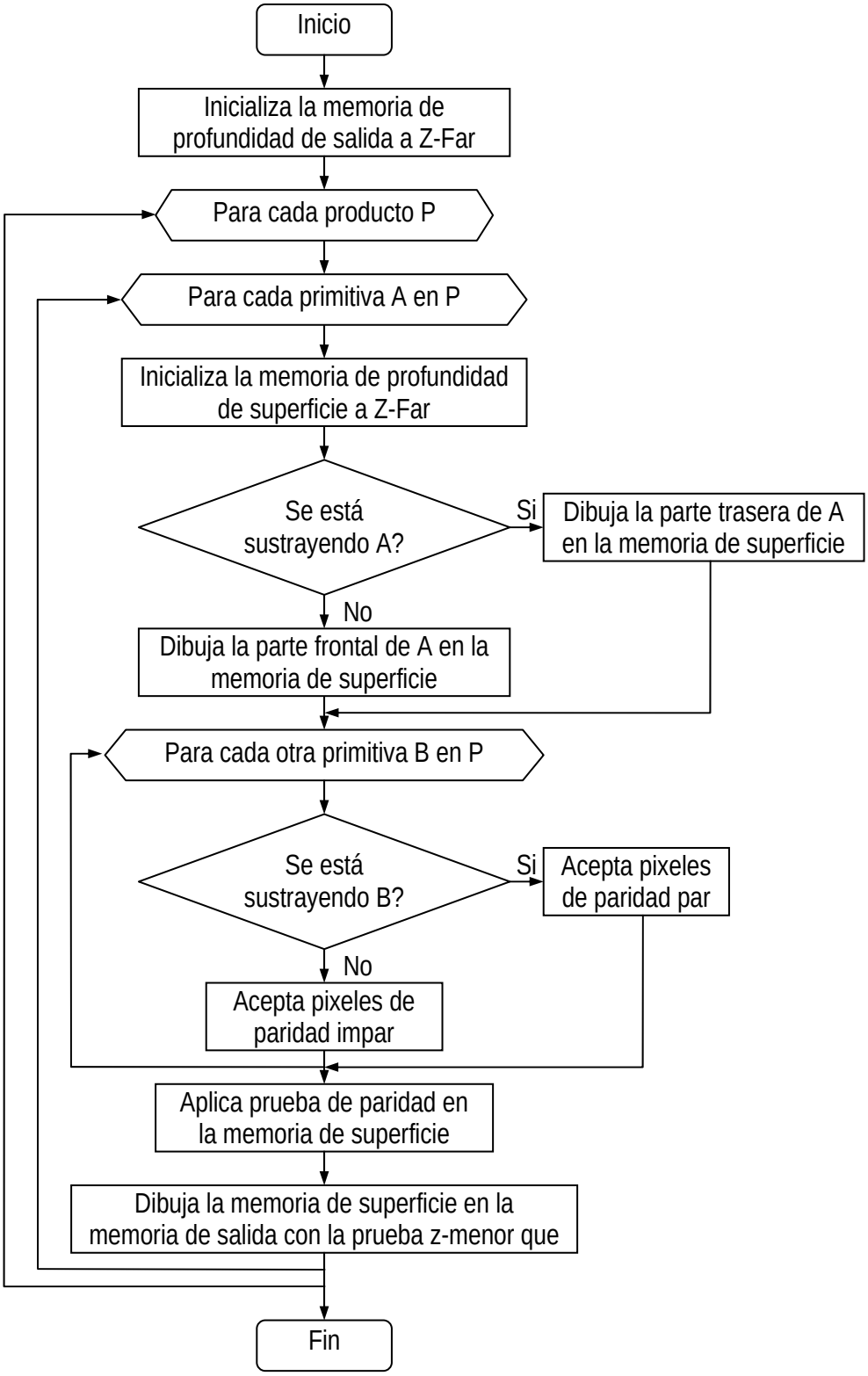


Figura 29. Diagrama de flujo del algoritmo de Goldfeather

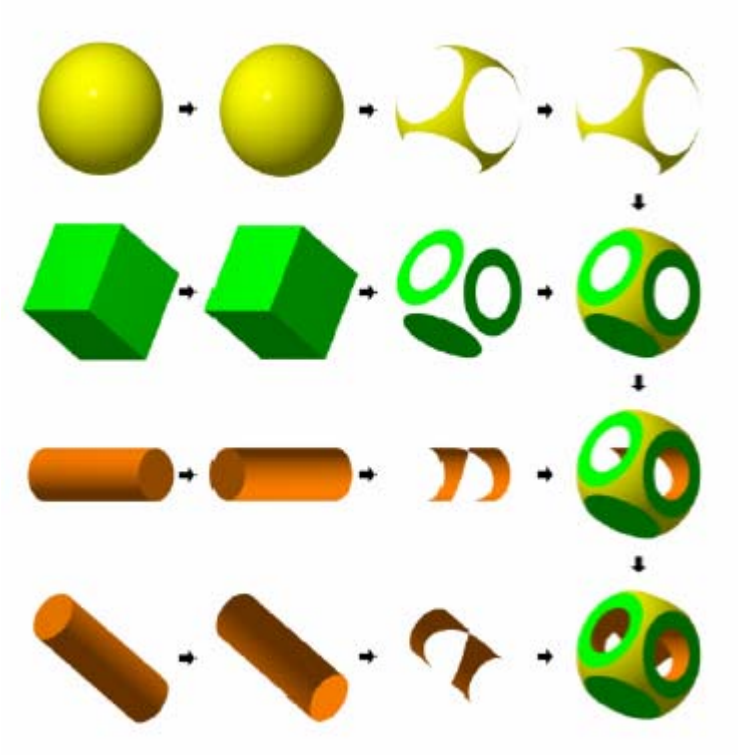


Figura 30. Ejemplo de aplicación del algoritmo Goldfeather [Stewart, 2000]

En la Figura 30 se presenta un ejemplo de aplicación del algoritmo de Goldfeather. Cada fila representa un paso del algoritmo. La primera columna muestra la primitiva seleccionada para recorte en el paso actual. En la segunda columna se muestra la superficie apropiada para ser dibujada ya sea superficie frontal, o posterior. En la tercera columna, se muestra el contenido de la memoria de superficie donde se almacena la superficie recortada por el resto de primitivas en el producto. El paso final es agregar la superficie recortada a la memoria de salida.

4.12.2. Algoritmo Trickle

El algoritmo Trickle propuesto en 1989 por D. Epstein, F. Jansen, y J. Rossignac [Epstein, 1989] substrae capas desde adelante hacia atrás con respecto al punto de vista. Se usan dos memorias de profundidad para ir cambiando a través de la secuencia de capas, en una memoria de profundidad se almacena el resultado acumulado y se usa también otra memoria de profundidad como almacenamiento temporal; en total se utilizan cuatro memorias de profundidad. En la Figura 31 se presenta este algoritmo en forma de diagrama de flujo.

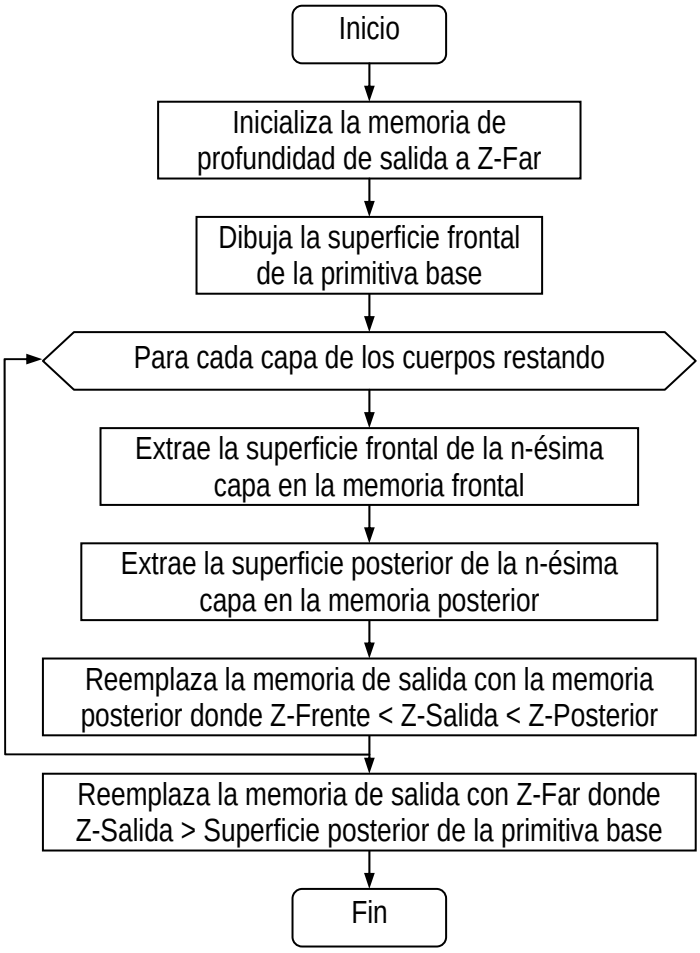


Figura 31. Diagrama de flujo del algoritmo Trickle

4.12.3. Algoritmo Goldfeather por capas

Como se mencionó anteriormente, Goldfeather y otros desarrollaron un algoritmo para la puesta en escena de modelos CSG de objetos convexos y otro para no convexos; Wiegand a su vez, presentó una variante del algoritmo de Goldfeather para hardware estándar. En 1998, Nigel Stewart, Geoff Leach y Sabu John [Stewart, 1998] introdujeron la idea de sacar provecho de la complejidad de profundidad del modelo y agrupar las primitivas por capas y realizar el recorte de superficies no por primitivas como Goldfeather lo plantea, sino por capas; de tal forma que el dibujado de primitivas en la memoria de superficie se haga no de una en una sino en grupos. El algoritmo propuesto se puede ver en el diagrama de flujo de la Figura 32. Se utilizan dos memorias de profundidad de manera similar al algoritmo de Goldfeather. La memoria de superficie se utiliza para sacar y recortar cada capa de superficies en el producto. La memoria de salida acumula el resultado final, tomando el pixel más cercano de cada elemento de la memoria de superficie. De forma semejante al algoritmo de Goldfeather, un pase final sobre todas las primitivas dibuja el color correcto en la memoria de recuadro, usando una prueba de “z-igual a” en la memoria de salida.

Desafortunadamente este algoritmo no funciona con todas las implementaciones de OpenGL, debido a la prueba final de profundidad. El problema se presenta por la transferencia de información entre la memoria de profundidad y la memoria estándar de la máquina y viceversa debido al redondeo de los datos; esto presenta problemas de inexactitud en la prueba de profundidad, la cual puede fallar. [Erhart, 2000]

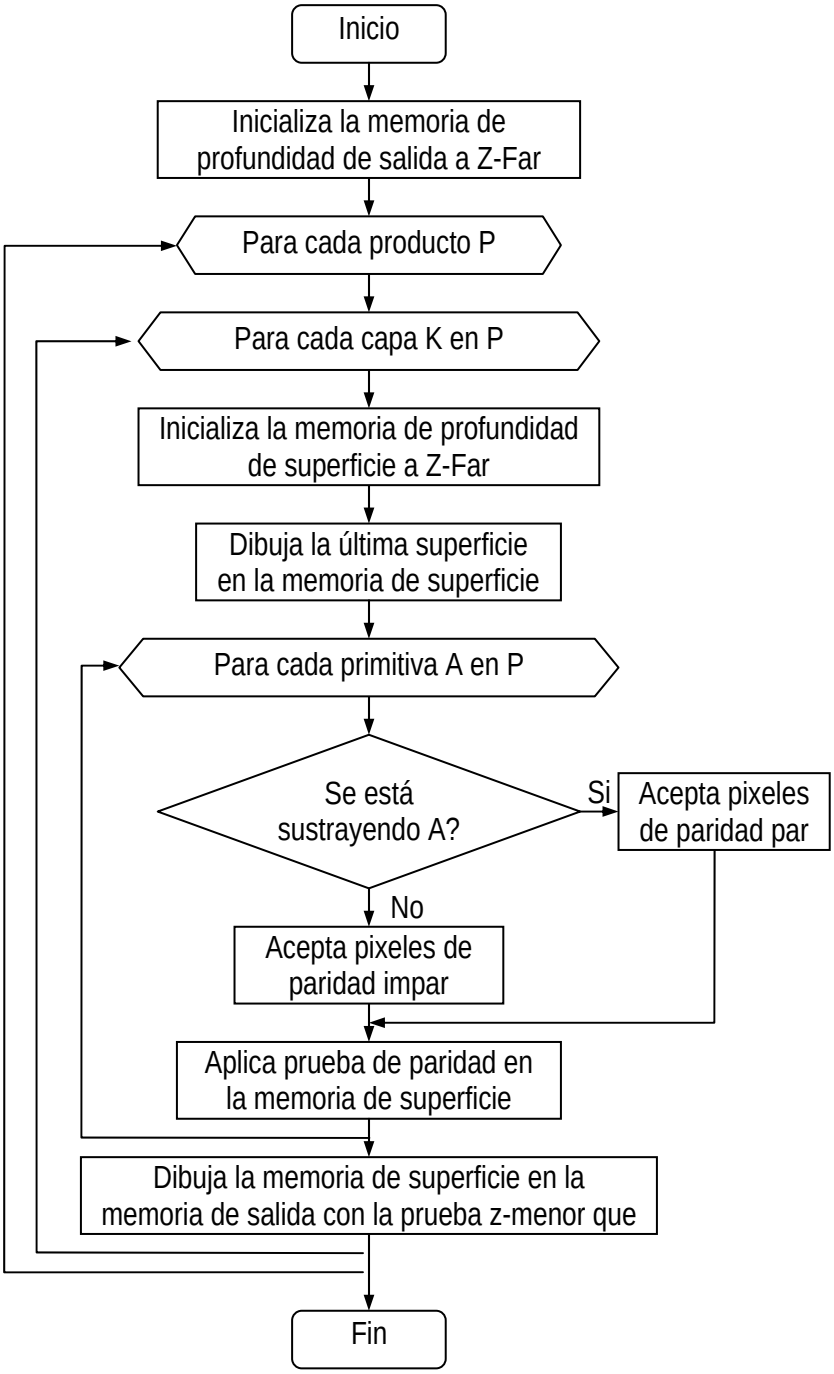


Figura 32. Diagrama de flujo del algoritmo Goldfeather por capas

4.12.4. Algoritmo Erhart-Tobler

En 2000 Günter Erhart y Robert F. Tobler [Erhart, 2000] propusieron una variante mejorada para el algoritmo Goldfeather por capas para soslayar el problema de fallas en la prueba de profundidad provocado al copiar desde la memoria de profundidad a la memoria estándar de la máquina y viceversa. Es posible usar este algoritmo para implementaciones con sistemas estándar OpenGL. Este algoritmo trabaja de forma similar al algoritmo Wiegand [Wiegand, 1996], en el que se almacenan máscaras binarias del resultado de recorte de superficies en la memoria de plantilla. En este algoritmo en lugar de almacenar una máscara por primitiva, se almacena una máscara por cada capa. De acuerdo a Erhart y Tobler, el número de máscaras de capas que pueden ser almacenadas simultáneamente en la memoria de plantilla depende de la complejidad de profundidad de la imagen. Ver la Tabla 3.

Tabla 3. Capas que se pueden almacenar simultáneamente en memoria de plantilla

Total de capas	Plantilla 4 bits	Plantilla 8 bits	Plantilla 16 bits
<= 2	2	2	2
<= 4	2	4	4
<= 8	1	5	8
<= 16		4	12
<= 32		3	11
<= 64		2	10
<= 128		1	9

Si la capacidad de la memoria de plantilla no es suficiente para almacenar todas las capas que se requiera, es entonces necesario hacer transferencias de información hacia la memoria estándar de la máquina; sin embargo aún tiene sus ventajas [Erhart, 2000].

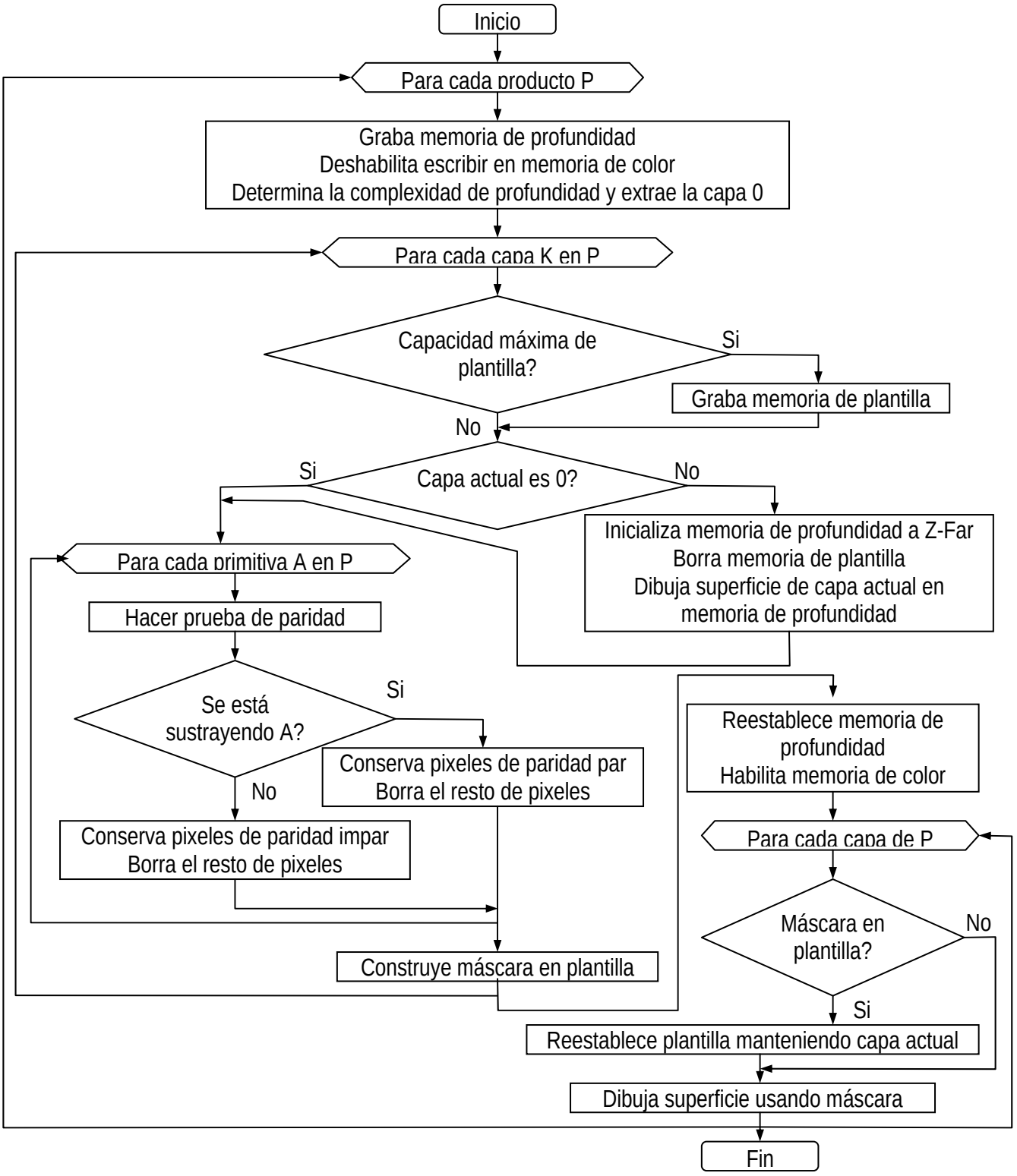


Figura 33. Diagrama de flujo del algoritmo Erhart-Tobler

4.12.5. Algoritmo SCS

En 2000 Nigel Stewart, Geoff Leach y Sabu John desarrollaron un algoritmo que le llamaron Sustracción Secuencial de Convexos SCS (Sequenced Convex Substraction) [Stewart, 2000] y luego una versión refinada en 2002 [Stewart, 2002]. Este algoritmo se basa en una substracción de objetos convexos en una secuencia específica. Este algoritmo trabaja en forma similar al algoritmo Trickle, en el que se extraen objetos convexos de la memoria; sin embargo, en lugar de extraer una secuencia ordenada de capas, las primitivas convexas se extraen en cualquier orden posible en dirección del punto de vista. En el mejor de los casos, no existe ningún pixel cubierto por más de una primitiva, y cualquier secuencia que cubra todas las primitivas será suficiente. En el peor de los casos, todas las primitivas se traslapan sobre un solo pixel y todos los n! escenarios se requieren. En forma interesante, todo esto se puede codificar con una secuencia de n^2 de longitud. [Stewart, 2000]

Se ha venido diciendo que este algoritmo trabaja para objetos convexos; sin embargo, también puede manejar objetos no convexos, si éstos se dividen en objetos convexos más sencillos.

En la Figura 34 se muestra un ejemplo de aplicación de este algoritmo. En la Figura 35 se muestra el algoritmo SCS usado específicamente para productos.

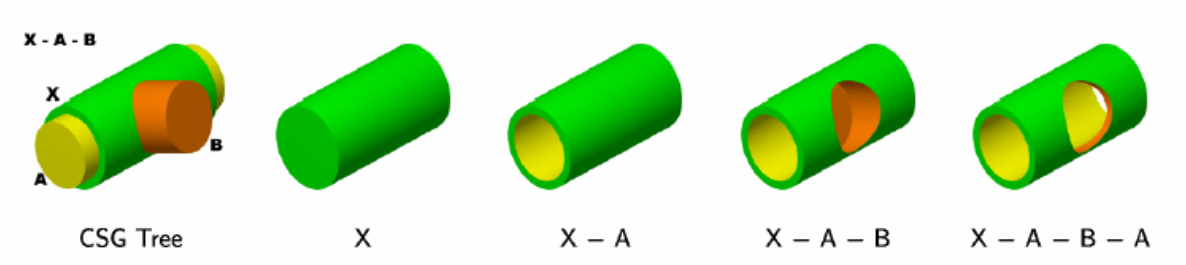


Figura 34. Ejemplo de aplicación del algoritmo SCS [Stewart, 2000]

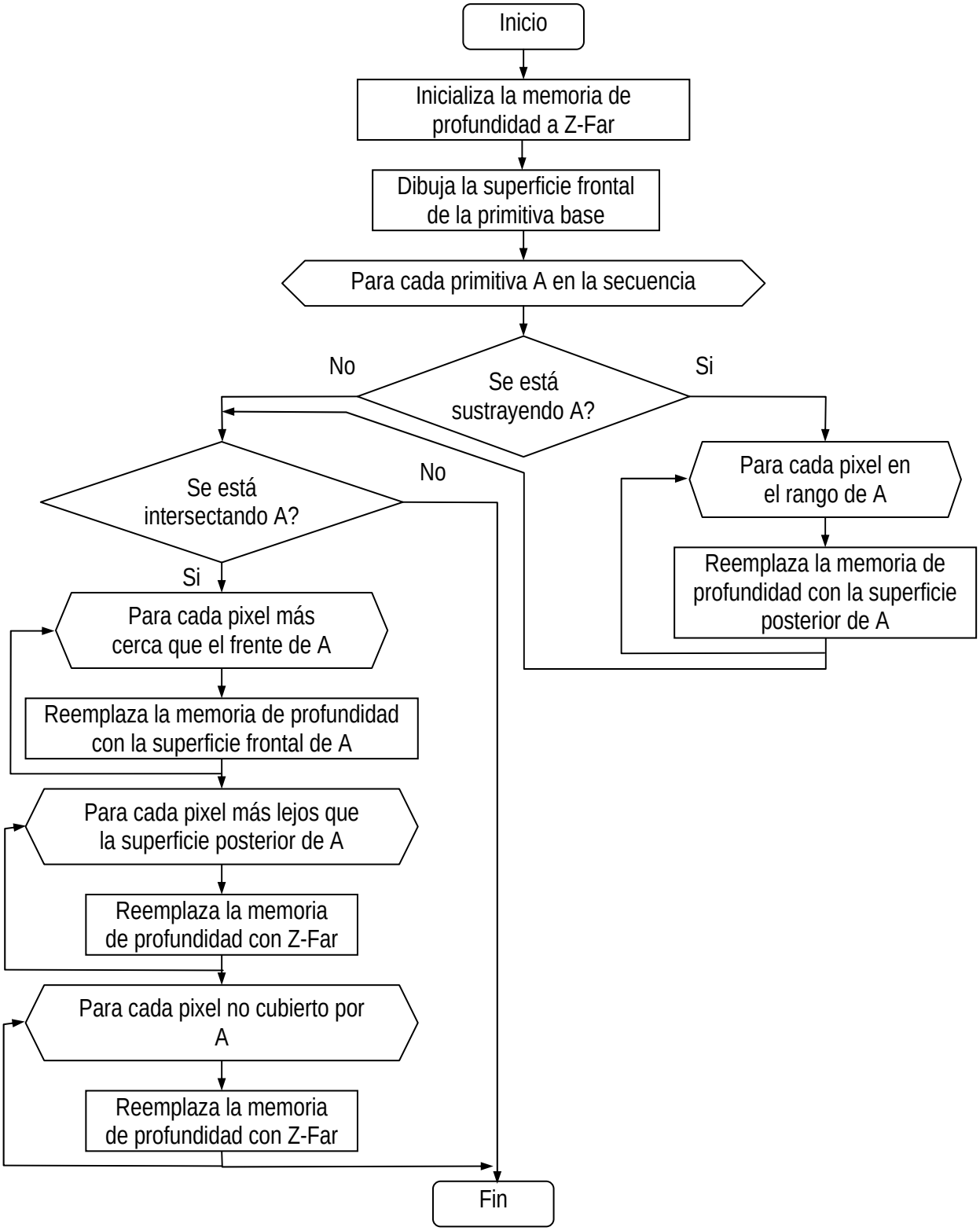


Figura 35. Diagrama de flujo del algoritmo SCS para productos

4.12.6. Algoritmo Guha

En 2003 Sudipto Guha, Shankar Krishnan, Kamesh Munagala y Suresh Venkatasubramanian como variante del algoritmo de Goldfeather, aplicaron la prueba de profundidad de ambos lados (esta prueba sólo es soportada por mapeo de sombras en hardware moderno), algo semejante al algoritmo Trickle. Su algoritmo aplica la prueba de paridad a las capas de profundidad de los productos parciales en un orden de adelante hacia atrás; de manera que la máscara de plantilla rechaza la actualización de vista de una capa CSG que previamente haya sido establecida. A este método se le ha llamado proceso de pelado por profundidad.

4.12.7. Algoritmo Wiegand

En 1996 Wiegand en su artículo “Interactive Rendering of CSG Models” [Wiegand, 1996] presentó un método para emular el algoritmo de Goldfeather que originalmente fue diseñado para los equipos con sistema Pixel-Powers; pero para hardware estándar. Las memorias de profundidad llamadas memoria de superficie, memoria de salida y memoria de imagen se emulan con pases múltiples a la memoria estándar de la máquina. Se considera que este método junto con el presentado por Tom McReynolds y David Blythe [McReynolds, 1998] son los más prácticos de los sugeridos a la fecha [Rappoport, 1997].

Este algoritmo requiere que el árbol CSG se encuentre normalizado. Para referencia en qué es un árbol CSG normalizado y cómo hacerlo, se puede consultar la sección 4.12.1 Algoritmo Goldfeather. El árbol CSG normalizado es un árbol binario; pero es importante pensar en el árbol como una suma de productos para entender el procedimiento de plantilla CSG.

Se debe considerar a todas las uniones como sumas y todas las intersecciones y substracciones como productos. Nótese que la substracción es equivalente a una intersección con el complemento del término de la derecha. Es decir, $A-B=A\cap\overline{B}$. Una vez teniendo el árbol CSG normalizado, expresado como una suma de ramales, donde cada ramal está compuesto por intersecciones y substracciones y que el ramal de la derecha de cada operación es siempre una primitiva y que el ramal de la izquierda es otra operación o una primitiva, entonces se puede considerar eliminar los términos redundantes de cada producto; es decir, si un término se resta de sí mismo $A-A$, se puede eliminar, y si un término se intersecta consigo mismo $A\cap A$, se puede reemplazar por sí mismo.

Todas las uniones se pueden dibujar encontrando las superficies visibles de los ramales izquierdo y derecho del árbol y permitiendo que la prueba de profundidad encuentre las superficies visibles. Todos los productos se pueden dibujar al encontrar las superficies visibles de cada primitiva en el producto y recortando aquellas superficies por los cuerpos de las otras primitivas en el producto. Por ejemplo, para dibujar $A-B$, las superficies visibles de A se recortan por el complemento del cuerpo B, y las superficies visibles de B se recortan por el cuerpo A.

Las superficies visibles de un producto son las caras frontales de los operandos de las intersecciones y las caras posteriores de los operandos de la derecha de las substracciones.

Los sólidos cóncavos se procesan como conjuntos de superficies frontales y posteriores. La “convexidad” de un objeto se define como el número par máximo de superficies frontales y posteriores que se pueden dibujar vistos desde un punto de vista determinado. La n-ésima superficie frontal de una primitiva k-convexa se denota por Λ_{nf} y la n-ésima superficie posterior es Λ_{nb}. Dado

que un sólido puede cambiar de convexidad dependiendo del punto de vista, una representación acertada de la convexidad de la primitiva puede ser difícil y puede requerir que se reevalué el árbol CSG cada vez que cambie el punto de vista. En su lugar, se debe proporcionar la convexidad máxima posible al algoritmo para dibujar la n-ésima superficie frontal de la primitiva usando un contador en los planos de plantilla.

Más adelante, se debe reducir el árbol CSG a una “suma de productos” convirtiendo cada producto a una unión de productos, en el que cada uno consista en el producto de las superficies visibles de la primitiva objeto con los términos remanentes en el producto.

Dado que el término objeto en cada producto se ha reducido a una superficie frontal o posterior, los volúmenes de entorno de dicho término será un subconjunto del volumen de entorno de la primitiva original. Una vez que el árbol se convierte a productos parciales, se puede aplicar una vez más el proceso de podado del árbol.

En cada ramal del árbol CSG resultante que representa un producto parcial, el término de la extrema izquierda es conocido como “objeto”, y los términos remanentes en los ramales de la derecha se les conoce como primitivas de recorte.

La suma resultante de los productos parciales reduce el problema de dibujado a dibujar cada producto parcial correctamente antes de dibujar la unión de los resultados. Cada producto parcial es dibujado al dibujar la superficie objeto del producto parcial y después clasificando los pixeles generados por tal superficie con los valores de profundidad generados por cada una de las primitivas de recorte en el producto parcial. Si los pixeles dibujados por las primitivas de recorte pasan la

prueba de profundidad un número par de veces, tal pixel en la primitiva objeto esta afuera y es descartado. Si el conteo es impar, el pixel está adentro y se conserva.

Dado que el algoritmo graba la memoria de profundidad entre cada objeto a procesar, se puede optimizar al grabar y restaurar si se utilizan tantas primitivas objeto y de recorte como quepan en la memoria de plantilla.

El algoritmo usa un bit de plantilla S_p como indicador si la prueba de profundidad pasa para la primitiva, una cantidad n de bits de plantilla para contabilizar la n-ésima superficie S_{count}, donde n es el menor número para el que 2^n sea mayor que la convexidad máxima del cuerpo actual, y tantos bits S_α como sea posible para acumular si los pixeles objeto tienen que ser descartados o no.

Una vez que el árbol sea convertido a una suma de productos parciales, los productos individuales son dibujados. Los productos se agrupan juntos de forma que se puedan dibujar tantos productos parciales entre grabar y restaurar la memoria de profundidad, como tenga capacidad la memoria de plantilla.

Para cada grupo, se deshabilita escribir sobre la memoria de color, se graba y se limpia el contenido de la memoria de profundidad. Entonces, cada primitiva objeto se clasifica contra sus primitivas de recorte. Se reestablece la memoria de profundidad y cada primitiva objeto se dibuja utilizando la plantilla de recorte. Otra optimización se puede lograr al grabar y restaurar la memoria de profundidad grabando y restaurando sólo la región que contenga las partes de la superficie objeto que va a ser proyectada en pantalla. La clasificación consiste en dibujar la profundidad de cada primitiva objeto y después limpiar aquellos valores en los que la primitiva objeto sea determinada

estar fuera de las primitivas de recorte. Los valores de profundidad de las superficies se dibujan al dibujar la primitiva que contiene la superficie objeto deshabilitando la escritura en las memorias de color y plantilla. Se usa el contador S_{count} para enmascarar todo el cuerpo con excepción de la superficie objeto. En la práctica, la mayoría de las primitivas son convexas, de manera que el algoritmo es apropiado para tal caso. Después cada primitiva se dibuja de acuerdo a su turno. Se habilita la prueba de profundidad y se deshabilita escribir sobre la memoria de profundidad. Las operaciones de plantilla se enmascaran a S_p y el bit S_p de la plantilla se pone a "0". Tanto la función como la operación de plantilla se ponen de forma que el bit S_p cambie su valor cada vez que se realice exitosamente la prueba de profundidad para cada fragmento de la primitiva de recorte. Después de dibujar la primitiva de recorte, si este bit es "0" para primitivas no complementarias (o "1" para primitivas complemento), el pixel objeto se encuentra fuera y se debe marcar para ser descartado, habilitando escribir sobre la memoria de profundidad y almacenando el valor de profundidad máxima Z_f en la memoria de profundidad cada vez que el bit S_p indique si se debe descartar. Ya que se termine de dibujar todas las primitivas de recorte, los valores en la memoria de profundidad son Z_f para todos los pixeles objeto marcados como "fuera". El bit S_α para la primitiva es puesto a "1" donde el valor de profundidad de un pixel no sea igual a Z_f y "0" en caso contrario. Cada primitiva objeto en el grupo finalmente es dibujada en la memoria de recuadro con la prueba de profundidad y escritura en la memoria de profundidad habilitadas, la memoria de color también habilitada, la función y operación de plantilla puestas para escribir la profundidad y el color donde pase la prueba de profundidad y el bit S_α sea "1". Solamente se dibujan los pixeles internos a las primitivas de recorte.

4.13. Vista 3D estereo

La vista 3d estereo es una técnica muy común para incrementar el realismo de las imágenes proyectadas en pantalla. Para lograr este efecto, se crean dos vistas de la misma imagen, una para el ojo de la izquierda y la otra para el ojo de la derecha. Se utiliza un tipo especial de hardware para poner en pantalla las dos vistas de la escena en pantalla, de manera que cada ojo puede ver solamente la vista que le corresponde. La profundidad aparente de los objetos es una función de la diferencia de sus posiciones desde cada ojo. Cuando se realiza adecuadamente, los objetos se miran como si tuvieran una profundidad real. Para animación, se utilizan las memorias auxiliares y estas se deben actualizar constantemente en cada cuadro. OpenGL tiene la capacidad de manejar las memorias auxiliares tanto para la derecha como para la izquierda para crear imágenes de este tipo.

4.14. Librerías auxiliares OpenGL

No obstante OpenGL provee muchas funciones que pueden ayudar en la simulación de escenas tridimensionales en pantalla, algunas funciones deseables para la modelación de sólidos y cuerpos en el espacio no están disponibles directamente en OpenGL; por ejemplo, para establecer la posición de la cámara o el punto de vista de una escena, en OpenGL es necesario calcular una matriz, lo que no para todos se facilita. Para facilitar las operaciones de modelación en OpenGL, existen las librerías auxiliares OpenGL. Estas librerías auxiliares incluyen GLU, GLUT, GLAUX y GLX.

4.14.1. Librería auxiliar GLU

La primera librería auxiliar a mencionar es GLU. Esta librería se ha convertido en un estándar y es parte del paquete cuando se instala la librería OpenGL. La estructura de esta librería incluye funciones más complejas, como por ejemplo para definir un cilindro o un disco, se requiere solamente un comando. También esta librería incluye funciones para trabajar con curvas spline, operaciones adicionales para cálculo de matrices y proyecciones.

4.14.2. Librería auxiliar GLUT

La siguiente librería auxiliar también ampliamente usada es GLUT. Realiza no solamente operaciones adicionales a la librería estándar OpenGL, sino también ofrece funciones para la operación con ventanas, teclado y ratón apuntador. Para trabajar con OpenGL en un sistema operativo del tipo de ventanas; ya sea Windows o X Windows, es necesario trabajar con una serie de ajustes preliminares para establecer parámetros que dependerán del sistema operativo concreto en que se quiera trabajar. Con la librería auxiliar GLUT, es mucho más fácil trabajar con dicho tipo de sistemas operativos, dado que para definir una ventana sólo se requiere de unos cuantos comandos; que se podrá trabajar interactivamente con el teclado y el ratón apuntador y sin depender del tipo de sistema operativo que se trate. La librería también ofrece algunas funciones, con las que se pueden definir figuras más complejas, como son conos o tetraedros; incluso con la ayuda de unos cuantos comandos se puede crear la figura de una tetera.

4.14.3. Librería auxiliar GLAUX

Hay otra librería semejante a la librería GLUT, que se conoce como librería GLAUX. Esta librería fue desarrollada por Microsoft Corporation para el sistema operativo Windows. Es muy semejante en funcionamiento y utilización a la librería GLUT, aunque es un poco menos poderosa. Otro inconveniente de esta librería es que es exclusivamente para uso en el sistema operativo Windows, mientras que GLUT puede ser usada en muchos otros sistemas operativos.

4.14.4. Librería auxiliar GLX

Existe otra librería auxiliar llamada GLX, que también agrega algunas funciones, pero que también está orientada a un sistema operativo en específico. Esta librería fue desarrollada para ser utilizada exclusivamente en el sistema operativo X Windows. Una de sus ventajas es que no solamente ofrece dibujado de cuerpos localmente sino también en forma remota a través de una red.

4.14.5. Librería Tipo OpenGL para Visual Basic

Para desarrollar programas para manejo de gráficos en computadora en Visual Basic, existen algunas posibilidades de métodos para proyectar las imágenes en pantalla; uno de ellos sería el desarrollar paso a paso la graficación de los objetos de la escena definiendo todas las funciones y procedimientos necesarios para ello; este tipo de programación es el descrito en el capítulo 3 de este trabajo. Otro método es utilizando la librería OpenGL; sin embargo, como se explicó en la introducción de este mismo capítulo, OpenGl en sí fue desarrollado para ser utilizado con lenguajes de programación C y C++. Sería posible utilizar las funciones de OpenGL si se declararan internamente en el código de programación en Visual Basic. Una mejor opción es utilizar la librería

tipo OpenGL de Patrice Scribe [VBOpenGL]; la cual soporta las especificaciones de OpenGL versión 1.2 [Segal, 1999] y puede manejar las funciones de la librería auxiliar GLUT versión 3.6 [Kilgard, 1996].

4.15. Alternativas OpenGL

OpenGL como librería para manejo de gráficos por computadora en aplicaciones profesionales como para el desarrollo de juegos de video, tiene también fuertes competidores; entre ellos se puede contar a Direct3D que es parte de la librería DirectX, desarrollada por Microsoft Corporation. Comparando estas dos librerías, de acuerdo a Roman Podobedov [Podobedov], es imposible decir si una de ellas es mejor o peor que la otra, dado que cada una de ellas tiene sus características y ventajas. Por ejemplo, si se comparan en el plan de transferencia de una plataforma a otra, la librería Direct3D trabaja solamente en plataformas Intel bajo el sistema operativo de Windows, mientras que es posible que programas escritos usando la librería OpenGL puedan ser transportados a diferentes plataformas, como Unix, Linux, SunOS, IRIX, Windows, MacOS y muchas otras.

Por otro lado, para programación orientada a objetos, se considera a Direct3D mejor que OpenGL; otra ventaja de Direct3D es que puede ser utilizada en hardware de bajo costo; mientras que OpenGL no puede ser utilizada en todas las tarjetas gráficas, sino más bien para aceleradores gráficos, OpenGL es el estándar por defecto. Además OpenGL es considerada como más fácil de entender y puede ser empleada para entrenamiento en manejo de gráficos tridimensionales.

Otra de las librerías ampliamente usada como estándar para programación de juegos de video ha sido Glide. Este estándar fue creado por 3Dfx Corporation exclusivamente para su uso en aceleradores de video Voodoo. GLide es más de bajo nivel que OpenGL y en comandos son muy

semejantes. GLide ya no es soportada por 3Dfx dado que ha pasado sus derechos a desarrolladores de software “open source” o de libre distribución.

Dentro de algunas otras librerías que existen, también se puede mencionar Heidi. Más que una librería es considerada un manejador (driver) para la visualización de escenas tridimensionales en 3D Studio Max sólo bajo Windows NT.

Cabe mencionar también que bajo la dirección de Microsoft y Silicon Graphics, se encuentra en actual desarrollo un proyecto llamado Fahrenheit, en el área de estandarización de gráficos por computadora.

5. Conceptos básicos de cinemática

En la representación de la cinemática de cuerpos en el espacio se pueden emplear diferentes modelos matemáticos aplicando los diferentes métodos que se han venido desarrollando a lo largo de la historia. Para el objetivo de este trabajo, es necesario hablar de multicuerpos rígidos; dado que prácticamente un brazo de robot como el CRS A465 se considera un conjunto de cuerpos rígidos enlazados en una cadena cinemática abierta. El hablar de cuerpos rígidos, significa que los cuerpos no cambian su tamaño, ni su forma, ni su consistencia cuando son afectados por fuerzas externas.

El brazo de robot; como cualquier otro cuerpo está sujeto a las leyes de la mecánica; las cuales se pueden expresar de diferentes formas. Por medio de la aplicación de un modelo matemático se puede conocer tanto la posición, velocidad y/o aceleración de cualquiera de sus componentes. El obtener o calcular ya sea la posición final del órgano efector o la posición de cada una de las articulaciones del brazo de robot se divide en dos problemas específicos; los que se les conoce como problema cinemático directo y problema cinemático inverso.

- ✓ Problema cinemático directo. Es aquel que conociendo los ángulos (en el caso de juntas rotacionales) o las longitudes de las extensiones (en caso de juntas prismáticas), así como las características físicas de la cadena cinemática, se obtiene la posición y orientación del extremo de la misma.

- ✓ Problema cinemático inverso. En forma análoga, el problema cinemático inverso es aquel que conociendo la posición y la orientación del extremo de la cadena cinemática, se obtiene la posición y orientación de cada una de las articulaciones.

La posición y orientación de un cuerpo rígido con respecto a un origen establecido en el espacio como se muestra en la Figura 35, pueden ser expresadas por la combinación de una traslación y una combinación de rotaciones.

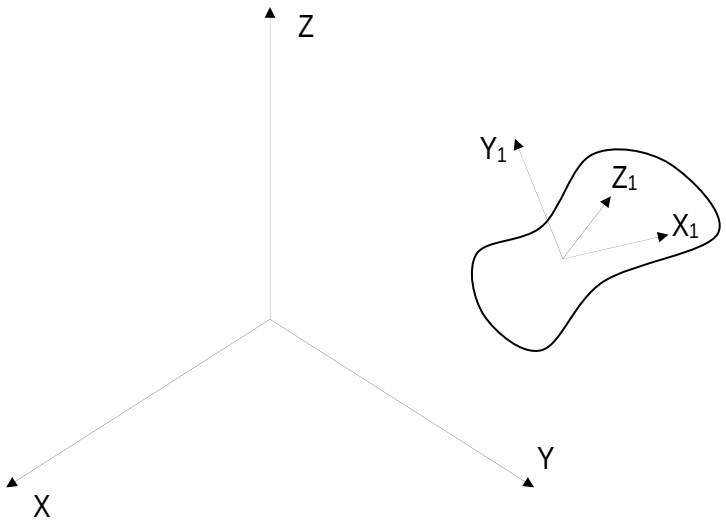


Figura 1. Definición de un cuerpo rígido en el espacio

5.1. Traslaciones

La traslación por un vector en $\mathbb{R}^3$, $v = [v_x, v_y, v_z]^T$ de un cuerpo rígido en el espacio euclidiano posicionado en el punto $p = [p_x, p_y, p_z]^T$, es simplemente el punto $p' = [p_x', p_y', p_z']^T$, tal que:

$$p' = p + v \tag{36}$$

es decir que,

$$\begin{bmatrix} p_x' \\ p_y' \\ p_z' \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} p_x + v_x \\ p_y + v_y \\ p_z + v_z \end{bmatrix} \tag{37}$$

La traslación también puede ser expresada en coordenadas homogéneas por medio del producto de una matriz de traslación por el vector de posición original, como se muestra en la siguiente ecuación:

$$p' = T_v p \tag{38}$$

Es decir,

$$\begin{bmatrix} p_x' \\ p_y' \\ p_z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{39}$$

5.2. Rotaciones

Rotación es un movimiento de un cuerpo de tal forma que al menos un punto dentro del mismo cuerpo permanece a la misma distancia a partir de una referencia fija. Se dice que al menos un punto permanece sin trasladarse cuando se está hablando en términos de rotaciones en dos dimensiones. Para el caso de rotaciones en tres dimensiones, puede ser más de un punto los que permanecen sin trasladarse y éstos forman una línea; a dicha línea se le conoce como eje de rotación.

Una propiedad importante de la rotación es que es una transformación lineal que no afecta la longitud de los vectores a rotar.

Sea un triángulo formado por los vectores a, b y c; de la forma mostrada en la Figura 37.

De la Figura 37, se puede ver que:

$$a = b - c \tag{40}$$

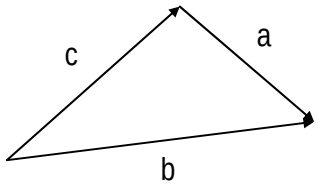


Figura 37. Triángulo formado por tres vectores.

utilizando el producto punto de los vectores en ambos lados de la identidad, se tiene que:

$$a \cdot a = (b - c) \cdot (b - c) \tag{41}$$

Y dado que:

$$(b - c) \cdot (b - c) = b \cdot b - 2b \cdot c + c \cdot c \tag{42}$$

Usando la definición del producto punto de la ecuación 17

$$\mathbf{A} \cdot \mathbf{B} = \left|\mathbf{A}\right|\left|\mathbf{B}\right| \cos\theta \tag{17}$$

Se puede deducir que:

$$\mathrm{A} \cdot \mathrm{A} = \left|\mathrm{A}\right|^2 \tag{43}$$

Dado que el ángulo formado es de 0°. A partir de la ecuación 43 y reordenando la ecuación 42, se tiene que:

$$b \cdot c = \frac{1}{2}\left(|b|^2 + |c|^2 - |a|^2\right) \tag{44}$$

Sustituyendo la ecuación 40 en la ecuación 43

$$b \cdot c = \frac{1}{2}\left(|b|^2 + |c|^2 - |b-c|^2\right) \tag{45}$$

Esta última expresión establece que el producto punto entre dos vectores de un espacio vectorial puede ser expresado en términos únicamente de las longitudes de los mismos vectores. Entonces, una rotación no solamente es una transformación lineal que conserva la longitud, sino que también conserva el ángulo entre los vectores a rotar.

La composición de dos o más rotaciones es también una rotación y cada rotación tiene una inversa única que es también una rotación.

Otra característica importante de las rotaciones es que forman un grupo no abeliano bajo la multiplicación; es decir, no cumplen con la propiedad conmutativa por lo que el orden de aplicación de varias rotaciones consecutivas afectan el resultado final.

Para el caso de las rotaciones, existen diferentes métodos para expresar su modelo matemático; entre ellos, se puede mencionar matrices de rotación, ángulos de Euler y quaterniones. Cada uno de ellos tiene sus ventajes y desventajas con respecto a los demás de acuerdo a su aplicación.

5.2.1. Matrices de rotación

Sea un punto P localizado en el espacio con las coordenadas tal que:

$$P = P_x i + P_y j + P_z k \tag{46}$$

donde i, j y k, son vectores unitarios a lo largo de los ejes X, Y y Z respectivamente de un sistema tridimensional estacionario de referencia. Y P_x, P_y y P_z son las componentes de localización o coordenadas del punto P.

Considerando también un segundo sistema tridimensional móvil $X'Y'Z'$, que se moverá conforme se mueva el punto a rotar, con sus ejes originalmente alineados a los ejes del sistema estacionario de referencia. De modo que el punto P con referencia a este segundo sistema será:

$$P = P_{x'} i' + P_{y'} j' + P_{z'} k' \tag{47}$$

donde i', j' y k', son ahora vectores unitarios a lo largo de los ejes X', Y' y Z' respectivamente de este segundo sistema de referencia. Y $P_{x'}$, $P_{y'}$ y $P_{z'}$ son las componentes de localización o coordenadas del punto P con respecto al mismo sistema.

Igualando las ecuaciones 46 y 47, se tiene:

$$P = P_X i + P_Y j + P_Z k = P_{X'} i' + P_{Y'} j' + P_{Z'} k' \tag{48}$$

Si se hace rotar el punto P un ángulo α a través del eje Z y junto con él se rota también el segundo sistema, se tiene entonces que los vectores unitarios a lo largo de los ejes del sistema secundario son:

$$i' = \cos\alpha\, i + \operatorname{sen}\alpha\, j \tag{49}$$

$$j' = -\text{sen}\alpha\, i + \cos\alpha\, j \tag{50}$$

$$k' = k \tag{51}$$

Sustituyendo las ecuaciones 49, 50 y 51 en la ecuación 47, se tiene:

$$P = P_{x'}(\cos\alpha\, i + \text{sen}\alpha\, j) + P_{y'}(-\text{sen}\alpha\, i + \cos\alpha\, j) + P_{z'}k \tag{52}$$

Desarrollando y reordenando, se tiene:

$$P = (P_{x'}\cos\alpha - P_{y'}\,\text{sen}\alpha)\, i + (P_{x'}\,\text{sen}\alpha + P_{y'}\cos\alpha)\, j + P_{z'}\, k \tag{53}$$

En forma de matriz, se tiene:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} \cos\alpha & \text{-sen}\alpha & 0 \\ \text{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_{x'} \\ P_{y'} \\ P_{z'} \end{bmatrix} \tag{54}$$

Esta ecuación se puede interpretar de la siguiente forma: teniendo el vector de coordenadas tridimensionales de un punto en el espacio se puede multiplicar por una matriz de rotación para rotar el punto a través del eje Z un ángulo α. La matriz de rotación para rotar en el eje Z es entonces:

$$R_z = \begin{bmatrix} \cos\alpha & -\text{sen}\alpha & 0 \\ \text{sen}\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{55}$$

Siguiendo un procedimiento semejante al anterior se puede encontrar las matrices de rotación sobre los ejes X y Y, las cuales son:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\text{sen}\alpha \\ 0 & \text{sen}\alpha & \cos\alpha \end{bmatrix} \tag{56}$$

$$R_y = \begin{bmatrix} \cos\alpha & 0 & \text{sen}\alpha \\ 0 & 1 & 0 \\ -\text{sen}\alpha & 0 & \cos\alpha \end{bmatrix} \tag{57}$$

Para el caso de rotaciones sobre un eje arbitrario, sea el caso que se quiere rotar un punto cualquiera en el espacio un ángulo (delta) δ con respecto a un eje arbitrario que por el origen y cuya reflexión sobre el plano XY forma un ángulo (alfa) α con respecto al eje X, y además forma un ángulo (beta) β entre el mismo eje de rotación y el plano XY, tal como se muestra en la Figura 38.

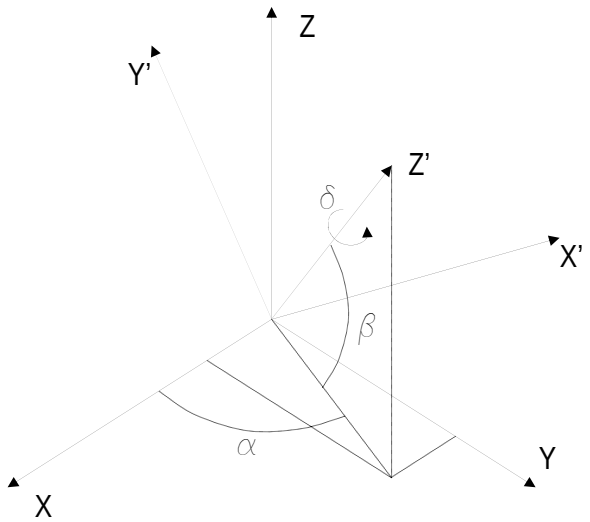


Figura 38. Rotación con respecto a un eje arbitrario.

La rotación del punto se logra con el siguiente proceso:

1. Siguiendo un procedimiento igual al descrito en el capítulo 3, se puede construir una matriz de transformación, de manera que al multiplicar un vector de localización del punto a rotar

por la matriz de la ecuación 14, se obtenga la representación del mismo punto con respecto a un sistema secundario $X'Y'Z'$.

2. Multiplicar por la matriz de rotación respecto al eje Z un ángulo (delta) δ de la ecuación 55.

3. Por último multiplicar por la matriz de transformación inversa para representar un punto desde el sistema secundario al sistema original (ecuación 7).

$$\begin{bmatrix} P_{x'} \\ P_{y'} \\ P_{z'} \end{bmatrix} = \begin{bmatrix} -\mathrm{s}\alpha & -c\alpha\,\mathrm{s}\beta & \mathrm{c}\alpha\,\mathrm{c}\beta \\ \mathrm{c}\alpha & -\mathrm{s}\alpha\,\mathrm{s}\beta & \mathrm{s}\alpha\,\mathrm{c}\beta \\ 0 & \mathrm{c}\beta & \mathrm{s}\beta \end{bmatrix} \begin{bmatrix} c\delta & -\mathrm{s}\delta & 0 \\ \mathrm{s}\delta & c\delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\mathrm{s}\alpha & c\alpha & 0 \\ -c\alpha\,\mathrm{s}\beta & -\mathrm{s}\alpha\,\mathrm{s}\beta & \mathrm{c}\beta \\ \mathrm{c}\alpha\,\mathrm{c}\beta & \mathrm{s}\alpha\,\mathrm{c}\beta & \mathrm{s}\beta \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad (58)$$

Los ángulos (alfa) α y (beta) β, se calculan tal como se describe en la sección 3. 3 Cálculo de los ángulos de localización del punto de vista.

Todas estas matrices de rotación vistas en esta sección forman un grupo bajo la operación de multiplicación. A este grupo se le conoce como grupo especial de matrices ortogonales con notación SO(n), para un espacio vectorial n-dimensional. Estas matrices cuyo determinante es igual a 1, tienen la característica que sus columnas; así como sus renglones son vectores unitarios mutuamente ortogonales. Para este grupo especial, el ser ortogonales implica también que la transpuesta de la matriz es igual a la inversa de la misma matriz.

5.2.2. Ángulos de Euler

De forma semejante a la rotación anterior con respecto a un eje cualquiera, de acuerdo al teorema de rotación de Euler, cualquier rotación puede ser descrita usando tres ángulos; que son los

llamados ángulos de Euler. Estos permiten fijar una orientación cualquiera en el espacio mediante la aplicación de tres rotaciones consecutivas de la siguiente forma:

1. Una rotación de ángulo ϕ con respecto al eje z

2. Una rotación de ángulo θ con respecto al eje y' (y' es el eje resultante del eje y aplicando la rotación anterior)

3. Una rotación de ángulo ψ con respecto al eje z'' (z'' es el eje resultante del eje z' aplicando la rotación anterior)

Si estas rotaciones se representan en forma matricial, entonces una rotación general $R_{\phi,\theta,\psi}$, puede ser escrita como una combinación de las rotaciones individuales descritas anteriormente; es decir, rotaciones con respecto a los ejes girados o modificados por las rotaciones anteriores y también por la combinación de rotaciones con respecto a los ejes originales [Rodríguez, 2004].

$$R_{\phi,\theta,\psi} = R_{z'',\psi} R_{y',\theta} R_{z,\phi} = R_{z,\phi} R_{y,\theta} R_{z,\psi} \tag{59}$$

En forma matricial,

$$R_{\phi,\theta,\psi} = \begin{bmatrix} c\phi & -s\phi & 0 \\ s\phi & c\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{60}$$

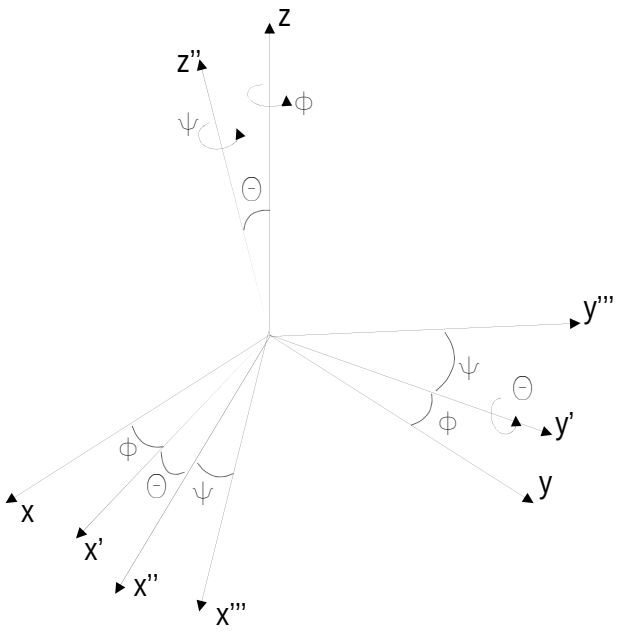


Figura 39. Ángulos de Euler.

Multiplicando,

$$R_{\phi,\theta,\psi} = \begin{bmatrix} c\theta\, c\phi\, c\psi - s\phi\, s\psi & -c\psi\, s\phi - c\theta\, c\phi\, s\psi & c\phi\, s\theta \\ c\theta\, c\psi\, s\phi + c\phi\, s\psi & c\phi\, c\psi - c\theta\, s\phi\, s\psi & s\theta\, s\phi \\ -c\psi\, s\phi & s\phi\, s\psi & c\phi \end{bmatrix} \tag{61}$$

En el medio de la programación de gráficos por computadora, es muy conocido que al utilizar este método se corre el riesgo de obtener lo que se conoce como Gimbal lock. Este fenómeno se presenta cuando dos ejes de rotación de un objeto quedan alineados en la misma dirección y se pierde uno de los tres grados de libertad porque el objeto no puede girar como se espera. La razón de ésto es porque el método evalúa los ejes en forma independiente en un orden establecido.

Como ejemplo de la aplicación de los ángulos de Euler en la historia, está la utilización de cardanes para soportar el movimiento de la plataforma inercial del módulo lunar del proyecto espacial Apolo 11. Esta plataforma inercial era la que le decía a la computadora central de navegación la forma en que se encontraba orientada la nave; estaba montada en el interior de un arreglo anidado de tres cardanes funcionando con juntas de rotación. El cardán exterior (OG Outer Gimbal) estaba montado sobre un eje sujeto a la nave, de forma que el ensamble completo era libre de girar sobre el eje del cardán exterior (OGA Outer Gimbal Axis). El cardán medio (MG Middle Gimbal) estaba montado sobre la parte interna del cardán exterior a 90º del eje del cardán exterior. De forma similar, el cardán interior cargaba la plataforma que estaba sujeta en puntos perpendiculares al eje del cardán medio (MGA Middle Gimbal Axis). Ver Figura 40.

Los ángulos de los ejes de los cardanes de montaje eran en realidad los ángulos de Euler entre la plataforma inercial o miembro estable y la base de navegación medidos con relación a la base de navegación misma.

Conforme la nave maniobraba en el espacio, los cardanes giraban de forma tal que mantenían la plataforma inercial en la misma orientación absoluta. Suponiendo que la plataforma y los cardanes tuvieran una posición inicial tal como se muestran en la Figura 40, se pudiera decir que si la nave maniobrara una rotación con respecto al eje del cardán exterior, el sistema no tendría problema en mantener la plataforma perfectamente alineada y los tres ejes permanecerían mutuamente perpendiculares sin importar qué tan grande fuera el ángulo de rotación.

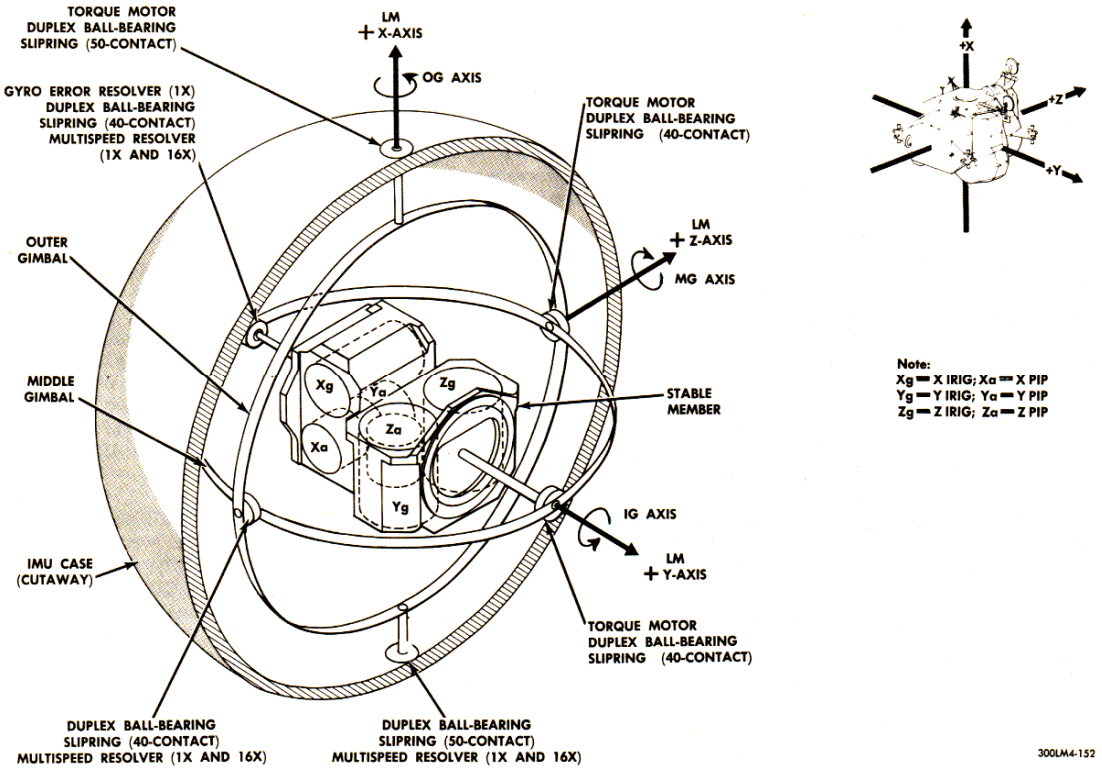


Figura 40. Montaje de la plataforma inercial del módulo lunar del proyecto Apolo 11.

Ahora, suponiendo de nuevo la posición original de acuerdo a la Figura 40, se pudiera decir también que si la nave maniobrara una rotación con respecto al eje del cardán interior, para cualquier magnitud de rotación el sistema mantiene los tres ejes perpendiculares mutuamente y sin problema la plataforma permanecería en su orientación original.

Sin embargo, por último suponiendo otra vez a partir de la orientación original, si se maniobrara una rotación con respecto al eje del cardán medio de forma que la parte superior del eje del cardán exterior se alejara hacia la parte posterior de la figura y que la parte inferior del mismo cardán se moviera hacia el frente de la figura. Entonces los ejes ya no permanecerían mutuamente perpendiculares y si la rotación efectuada fuera de 90º, entonces se tendrían los ejes de los

cardanes interior y exterior alineados y en esencia se tendría un sistema de dos ejes de rotación en lugar de tres localizados sobre un plano horizontal. Si después de esto se quisiera realizar una rotación sobre un eje perpendicular al plano horizontal mencionado, entonces se estaría rotando también la plataforma; por lo que ya no permanecería en orientación estable.

La solución dada a este problema del proyecto se nota en el registro de la conversación entre Owen Garriott en Houston y el piloto Mike Collins cerca de dos horas después de llegar a la luna.

-104:59:27 Garriott: Columbia, Houston. Over.

-104:59:34 Collins: Columbia. Go.

-104:59:35 Garriott: Columbia, Houston. We noticed you are maneuvering very close to gimbal lock. I suggest you move back away. Over.

-104:59:43 Collins: Yeah. I am going around it, doing a CMC Auto maneuver to the Pad values of roll 270, pitch 101, yaw 45.

-104:59:52 Garriott: Roger, Columbia. (Long Pause)

-105:00:30 Collins: (Faint, joking) How about sending me a fourth gimbal for Christmas. [Apollo 11]

La solución se explica en el documento E-1344 [Hoag, 1963] que consiste en instalar un cuarto cardán; llamado cardán redundante. El cual agrega un grado de libertad más a los tres teóricamente necesarios.

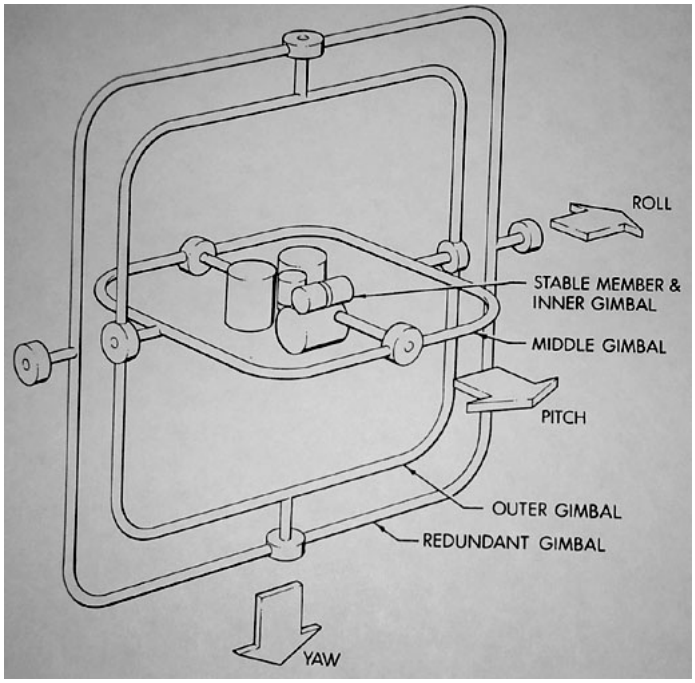


Figura 41. Plataforma inercial del módulo lunar del proyecto Apolo 11 con cuatro cardanes.

5.2.3. Quaterniones

La invención de los quaterniones ha sido atribuida al matemático irlandés William Rowan Hamilton (1805-1865). Hamilton al conocer el álgebra y aplicaciones de los números complejos y sabiendo que éstos eran aplicables solamente a la geometría plana o bidimensional, comenzó a trabajar en el desarrollo de un sistema de números hipercomplejos aplicables al espacio tridimensional y tratando de encontrar una forma de realizar productos y divisiones entre los elementos de este conjunto encontró que era imposible tratar de hacerlo con un sistema de triadas de números o números en tres dimensiones. Trabajando con tríadas incluyendo una parte real y dos partes imaginarias $\mathrm{T} = \mathrm{a} + i\mathrm{b} + j\mathrm{c}$, donde $\mathrm{a}, \mathrm{b}, \mathrm{c} \in \mathbb{R}$, encontró que era necesario considerar un sistema numérico de cuatro dimensiones y además que tenía que sacrificar la conmutatividad en la multiplicación para que

fuera posible la definición de la multiplicación. La palabra quaternión propiamente significa “un conjunto de cuatro” [Guthrie, 1886].

Los quaterniones se consideran como una extensión de los números complejos y predecesores del álgebra de vectores. [Shoemake, 1985]

Hamilton en [Hamilton, 1844-1850] definió los quaterniones como

$$Q = w + ix + jy + kz \tag{62}$$

de donde $w, x, y, z \in \mathbb{R}$ y además i, j y k son símbolos de tres cantidades imaginarias unitarias que no se encuentran linealmente relacionadas una con otra.

Otra forma de representar los quaterniones es:

$$Q = [w, x, y, z] \tag{63}$$

En [Palomares, 2006] se ha demostrado que el conjunto de los quaterniones, denotado por $H = \{[w, x, y, z] \mid w, x, y, z \in \mathbb{R}\}$ forma una estructura de campo no conmutativo bajo la multiplicación y que bajo las operaciones de suma y multiplicación por escalares forma un espacio vectorial real, por lo que se puede definir una transformación lineal que permite transformar los elementos de este espacio vectorial a elementos del espacio vectorial de $\mathbb{R}^3$ y viceversa; de forma que se puede realizar una representación geométrica de los cuerpos a modelar. Es decir, se puede realizar una representación con vectores de $\mathbb{R}^3$ de los cuerpos que conforman el robot a modelar, transformar dichos vectores a quaterniones, realizar operaciones de rotación y por último transformar el resultado de regreso a elementos de $\mathbb{R}^3$ para ser posteriormente proyectados en la pantalla.

Existe un subconjunto de quaterniones $\mathrm{Re}\mathbb{H}$, tal que $\mathrm{Re}\mathbb{H} = \{[w,0,0,0] \mid w \in \mathbb{R}\} \in \mathbb{H}$, al cual se le llama quaterniones puramente reales o quaterniones escalares. Se puede definir una transformación lineal que permita transformar los elementos de este subespacio vectorial a elementos de los números reales $\mathbb{R}$, de la forma:

$$T_R([w,0,0,0]) = w \ \forall \ [w,0,0,0] \in \mathrm{Re}\mathbb{H}, \text{donde } w \in \mathbb{R} \tag{64}$$

Se puede también definir una transformación inversa a 64, de forma que permita expresar los elementos de los números reales $\mathbb{R}$ como elementos del subespacio vectorial de los quaterniones puramente reales $\mathrm{Re}\mathbb{H}$, de la forma:

$$T_{\mathrm{Re}\mathbb{H}}(w) = [w,0,0,0] \ \forall \ w \in \mathbb{R}, \text{donde } [w,0,0,0] \in \mathrm{Re}\mathbb{H} \tag{65}$$

Así mismo, existe también un subconjunto de quaterniones $\mathrm{Im}\mathbb{H}$, tal que $\mathrm{Im}\mathbb{H} = \{[0,x,y,z] \mid x,y,z \in \mathbb{R}\} \in \mathbb{H}$, al cual se le llama a su vez quaterniones puramente imaginarios o quaterniones vectores. Se puede también definir la transformación lineal que permita transformar los elementos de este subespacio vectorial a elementos del espacio vectorial tridimensional $\mathbb{R}^3$, de la forma:

$$T_V([0,x,y,z]) = [x,y,z] \ \forall \ [0,x,y,z] \in \mathrm{Im}\mathbb{H}, \text{donde } x,y,z \in \mathbb{R} \tag{66}$$

Se puede también definir una transformación inversa a 66, de forma que permita expresar los elementos del espacio vectorial tridimensional $\mathbb{R}^3$, como elementos del subespacio vectorial de los quaterniones puramente imaginarios o quaterniones vectores $\mathrm{Im}\mathbb{H}$, de la forma:

$$T_{\mathrm{Im}\mathbb{H}}([x,y,z]) = [0,x,y,z] \ \forall \ [x,y,z] \in \mathbb{R}^3, \text{donde } [0,x,y,z] \in \mathrm{Im}\mathbb{H} \text{ y } x,y,z \in \mathbb{R} \quad (67)$$

5.2.3.1. Suma de quaterniones

Sea un quaternión de la forma expresada en la sección anterior y un segundo quaternión de la forma

$$\mathrm{Q}' = \mathrm{w}' + i\mathrm{x}' + j\mathrm{y}' + k\mathrm{z}' \quad (68)$$

Los cuales son elementos del conjunto $\mathbb{H}$ de los quaterniones.

El hablar de igualdad entre estos dos quaterniones $\mathrm{Q} = \mathrm{Q}'$, debe suponer que existe una igualdad uno a uno entre sus elementos; de forma que:

$$\mathrm{w} = \mathrm{w}', \ \mathrm{x} = \mathrm{x}', \ \mathrm{y} = \mathrm{y}', \ \mathrm{z} = \mathrm{z}' \quad (69)$$

se define entonces que la suma o resta de quaterniones tiene la forma:

$$\mathrm{Q} \pm \mathrm{Q}' = \mathrm{w} \pm \mathrm{w}' + i(\mathrm{x} \pm \mathrm{x}') + j(\mathrm{y} \pm \mathrm{y}') + k(\mathrm{z} \pm \mathrm{z}') \quad (70)$$

5.2.3.2. Multiplicación por escalares

Sea $\alpha \in \mathbb{R}$ y $\mathrm{Q} = \mathrm{w} + i\mathrm{x} + j\mathrm{y} + k\mathrm{z} \in \mathbb{H}$, la multiplicación $\alpha \cdot \mathrm{Q} \in \mathbb{H}$ se define como

$$\alpha \cdot \mathrm{Q} = \alpha\mathrm{w} + i\alpha\mathrm{x} + j\alpha\mathrm{y} + k\alpha\mathrm{z} \quad (71)$$

5.2.3.3. Producto de quaterniones

La multiplicación de los dos quaterniones de la ecuación 64 y la ecuación 68 se define como:

$$\begin{aligned} QQ' = {} & ww' + iwx' + jwy' + kwz' \\ & + ixw' + i^2xx' + ijxy' + ikxz' \\ & + jyw' + jiyx' + j^2yy' + jkyz' \\ & + kzw' + kizx' + kjzy' + k^2zz' \end{aligned} \tag{72}$$

Antes de llegar a una expresión del tipo:

$$QQ' = Q'' = w'' + ix'' + jy'' + kz'' \tag{73}$$

Es necesario definir qué pasa con los complejos unitarios multiplicados entre ellos mismos; para lo cual a Hamilton el 16 de Octubre de 1843 se le ocurrió escribir en el puente de Brougham lo siguiente:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{74}$$

Lo cual, por supuesto implica que:

$$ij = k, \;\; jk = i, \;\; ki = j \tag{75}$$

Y también que:

$$ji = -k, \;\; kj = -i, \;\; ik = -j \tag{76}$$

Entonces:

$$ji = -ij, \;\; kj = -jk, \;\; ki = -ik \tag{77}$$

Lo cual nos dice claramente que el producto de dos quaterniones no es conmutativo.

Sustituyendo las ecuaciones 74, 75, 76 y 77 en la ecuación 72, se encuentra que la multiplicación de dos quaterniones $Q = w + ix + jy + kz$ y $Q' = w' + ix' + jy' + kz'$, de la forma QQ', es:

$$\begin{aligned} QQ' = {} & ww' + iwx' + jwy' + kwz' \\ & + ixw' - xx' + kxy' - jxz' \\ & + jyw' - kyx' - yy' + iyz' \\ & + kzw' + jzx' - izy' - zz' \end{aligned} \tag{78}$$

Y reacomodando, se tiene:

$$\begin{aligned} QQ' = {} & ww' - xx' - yy' - zz' \\ & + i(wx' + xw' + yz' - zy') \\ & + j(wy' + yw' + zx' - xz') \\ & + k(wz' + zw' + xy' - yx') \end{aligned} \tag{79}$$

Entonces los elementos del quaternión producto son:

$$\begin{aligned} w'' &= ww' - xx' - yy' - zz' \\ x'' &= wx' + xw' + yz' - zy' \\ y'' &= wy' + yw' + zx' - xz' \\ z'' &= wz' + zw' + xy' - yx' \end{aligned} \tag{80}$$

Hamilton también expuso que los quaterniones pueden ser interpretados como la suma de una parte puramente real y una puramente imaginaria. Y esta parte imaginaria puede representar las coordenadas rectangulares de un punto en el espacio, como un vector de $\mathbb{R}^3$ [Hamilton, 1844-1850]. En [Palomares, 2006] en la sección $2.3.3.\ Quaterniones\ y\ vectores$, se expone también este tipo de notación. Otra forma de expresar el producto de dos quaterniones $Q = w + ix + jy + kz$ y $Q' = w' + ix' + jy' + kz'$, es en base a esta notación.

Para encontrar una nueva expresión del producto de los dos quaterniones, primero se considerará que los quaterniones son puramente imaginarios; es decir su parte real es igual a 0. De manera que:

$$\mathrm{A} = 0 + A_X i + A_Y j + A_Z k \tag{81}$$

$$\mathrm{B} = 0 + B_X i + B_Y j + B_Z k \tag{82}$$

Expresando el producto de los dos quaterniones a partir de la ecuación 78, se tiene que:

$$\begin{aligned}\mathrm{AB} = &-(A_X B_X + A_Y B_Y + A_Z B_Z) \\ &+ i(A_Y B_Z - A_Z B_Y) - j(A_X B_Z - A_Z B_X) + k(A_X B_Y - A_Y B_X)\end{aligned} \tag{83}$$

Comparando con las definiciones de producto punto y producto cruz de las ecuaciones 16 y 21,

$$\mathrm{A} \times \mathrm{B} = \begin{vmatrix} i & j & k \\ A_X & A_Y & A_Z \\ B_X & B_Y & B_Z \end{vmatrix} = (A_Y B_Z - A_Z B_Y) i - (A_X B_Z - A_Z B_X) j + (A_X B_Y - A_Y B_X) k \tag{16}$$

$$\mathbf{A} \cdot \mathbf{B} = A_X B_X + A_Y B_Y + A_Z B_Z \tag{21}$$

Se puede escribir el producto de los dos quaterniones puramente imaginarios como:

$$\mathrm{AB} = -\mathrm{A} \cdot \mathrm{B} + \mathrm{A} \times \mathrm{B} \tag{84}$$

Ahora considerando que los quaterniones a multiplicar son de la forma $\mathrm{A} = A_W + A_X i + A_Y j + A_Z k$ y $\mathrm{B} = B_W + B_X i + B_Y j + B_Z k$. Usando la definición para el producto de dos quaterniones de la ecuación 76 y reorganizando el resultado, se tiene:

$$\begin{aligned}\mathrm{AB} &= A_W B_W - A_X B_X - A_Y B_Y - A_Z B_Z \\ &+ A_W\left(B_X i + B_Y j + B_Z k\right) + B_W\left(A_X i + A_Y j + A_Z k\right) \\ &+ \left(A_Y B_Z - A_Z B_Y\right)i - \left(A_X B_Z - A_Z B_X\right)j + \left(A_X B_Y - A_Y B_X\right)\end{aligned} \tag{85}$$

Considerando los quaterniones como la suma de una parte real con un vector. $\mathrm{A} = A_W + A_V$, donde $A_V = A_X i + A_Y j + A_Z k$ y $\mathrm{B} = B_W + B_V$, donde $B_V = B_X i + B_Y j + B_Z k$, entonces:

$$\mathrm{AB} = A_W B_W - \mathrm{A}\cdot\mathrm{B} + A_W B_V + B_W A_V + \mathrm{A}\times\mathrm{B} \tag{86}$$

De esta forma se tiene una parte real igual a $A_W B_W - \mathrm{A}\cdot\mathrm{B}$ y una parte vectorial igual a $A_W B_V + B_W A_V + \mathrm{A}\times\mathrm{B}$.

5.2.3.4. Conjugado de un quaternión

El conjugado de un quaternión del tipo $\mathrm{Q} = \mathrm{w} + i\mathrm{x} + j\mathrm{y} + k\mathrm{z}$, a semejanza de los números complejos, se define por:

$$\mathrm{Q}^* = \mathrm{w} - i\mathrm{x} - j\mathrm{y} - k\mathrm{z} \tag{87}$$

De manera que si se efectúa el producto entre un quaternión y su conjugado:

$$\mathrm{QQ}^* = \left(\mathrm{w} + i\mathrm{x} + j\mathrm{y} + k\mathrm{z}\right)\left(\mathrm{w} - i\mathrm{x} - j\mathrm{y} - k\mathrm{z}\right) \tag{88}$$

Aplicando la definición de producto de quaterniones de la ecuación 77, se tiene:

$$\begin{aligned} QQ^{*} &= ww + xx + yy + zz \\ &+ i(-wx + xw - yz + zy) \\ &+ j(-wy + yw - zx + xz) \\ &+ k(-wz + zw - xy + yx) \end{aligned} \tag{89}$$

Dado que los elementos de ambos quaterniones son reales, la expresión se reduce a:

$$QQ^{*} = w^{2} + x^{2} + y^{2} + z^{2} \tag{90}$$

5.2.3.5. Inverso de un quaternión

El inverso multiplicativo de un quaternión de la forma $Q = w + ix + jy + kz$, es denotado por Q^{-1} y es aquel que:

$$QQ^{-1} = 1 \tag{91}$$

Dividiendo la ecuación 88 por sí misma,

$$\frac{QQ^{*}}{QQ^{*}} = 1 \tag{92}$$

Separando términos:

$$Q\left(\frac{Q^{*}}{QQ^{*}}\right) = 1 \tag{93}$$

Comparando con la ecuación 91 y sustituyendo con las ecuaciones 87 y 90, se tiene:

$$Q^{-1} = \frac{Q^{*}}{QQ^{*}} = \left(\frac{w - ix - jy - kz}{w^{2} + x^{2} + y^{2} + z^{2}}\right) \tag{94}$$

5.2.3.6. Producto punto de quaterniones

A semejanza del producto punto entre dos vectores, definido en la sección 3.3.2 Producto interno de dos vectores, siendo los quaterniones de la forma:

$$\mathrm{A} = \left(A_W + iA_X + jA_Y + kA_Z\right) \tag{95}$$

Y

$$\mathrm{B} = \left(B_W + iB_X + jB_Y + kB_Z\right) \tag{96}$$

el producto punto entre los dos quaterniones se define como:

$$\mathrm{A} \cdot \mathrm{B} = A_W B_W + A_X B_X + A_Y B_Y + A_Z B_Z \tag{97}$$

5.2.3.7. Norma de un quaternión

De forma semejante a un vector en $\mathbb{R}^2$ o en $\mathbb{R}^3$, la norma de un quaternión $\mathrm{A} = \left(A_W + iA_X + jA_Y + kA_Z\right)$ se define por:

$$\left\|\mathrm{A}\right\| = \sqrt{A_W^2 + A_X^2 + A_Y^2 + A_Z^2} \tag{98}$$

Lo cual es una consecuencia del teorema de Pitágoras.

Otra forma de representar la norma de un quaternión se obtiene también en forma semejante al plano complejo; como la raíz cuadrada del producto de un quaternión con su conjugado

$$\left\|\mathrm{A}\right\| = \sqrt{\mathrm{A}\mathrm{A}^*} \tag{99}$$

O bien, usando la definición de producto punto de dos quaterniones de 97, la norma de un quaternión se puede expresar como la raíz cuadrada del producto punto de un quaternión consigo mismo

$$\|\mathrm{A}\| = \sqrt{\mathrm{A} \cdot \mathrm{A}} \tag{100}$$

5.2.3.8. Quaternión unitario

En particular, un quaternión unitario es aquel cuya magnitud o norma es igual a 1. Cualquier quaternión puede ser normalizado al dividirse por su propia magnitud o norma.

Como se vio anteriormente, un quaternión cualquiera puede ser expresado como la suma de una parte real y una parte vectorial $\mathrm{Q} = U_R Q_W + U_V Q_V$, donde $U_R = [1,0,0,0]$ es un quaternión unitario puramente real y es un quaternión unitario puramente imaginario paralelo a la parte vectorial de Q, haciendo $Q_W = \cos\phi$ y $Q_V = \mathrm{sen}\phi$, entonces:

$$\mathrm{Q} = \cos\phi + U_V \mathrm{sen}\phi \tag{101}$$

Para comprobar si este quaternión es en realidad un quaternión unitario, se puede calcular su norma y elevarla al cuadrado. De la ecuación 99 se tiene que:

$$\|\mathrm{Q}\|^2 = \mathrm{Q}\mathrm{Q}^* \tag{102}$$

Sustituyendo la ecuación 101 en la ecuación 102, se tiene:

$$\|\mathrm{Q}\|^2 = (\cos\phi + U_V \mathrm{sen}\phi)(\cos\phi - U_V \mathrm{sen}\phi) \tag{103}$$

Que realizando el producto término a término:

$$\|\mathrm{Q}\|^2 = \cos^2\phi - U_V \cos\phi \,\mathrm{sen}\phi + U_V \cos\phi \,\mathrm{sen}\phi + U_V U_V^* \mathrm{sen}^2\phi \quad (104)$$

Tomando en cuenta que U_V es un quaternión unitario y como se vio en 99, entonces se puede decir que:

$$\|\mathrm{Q}\|^2 = \cos^2\phi + \mathrm{sen}^2\phi = 1 \quad (105)$$

Para el caso particular de los quaterniones unitarios, de 94 se puede deducir que:

$\mathrm{Q}^* = \mathrm{Q}^{-1}$ sí y sólo si $\|\mathrm{Q}\| = 1$ (106)

Un quaternión unitario representa un punto en el espacio 4-dimensional a una distancia de 1 a partir del origen. El conjunto de todos los quaterniones unitarios forman una 3-esfera de radio 1, también conocida como hiperesfera unitaria S^3; es decir, la esfera formada por todos los puntos equidistantes (a una distancia de 1) a partir del origen en un espacio 4-dimensional. A esta 3-esfera comúnmente se le conoce en inglés como glome, derivado del latín “glomus” que significa “bola”. Así como una 2-esfera (la esfera tridimensional) es la superficie bidimensional o plana formada por la esfera tridimensional, una 3-esfera es un objeto tridimensional que engloba el hipervolumen de una esfera en $\mathbb{R}^4$. De una manera más cruda, un glome es a una esfera lo que la esfera es a un círculo.

5.2.3.9. Quaterniones paralelos y perpendiculares

Sean los quaterniones $\mathrm{P} = P_W + iP_X + jP_Y + kP_Z$ y $\mathrm{Q} = Q_W + iQ_X + jQ_Y + kQ_Z \in \mathbb{H}$, se dice que estos dos quaterniones son paralelos $\mathrm{P} \| \mathrm{Q}$, si sus partes vectoriales, o dicho de otra forma, al

ser considerados como quaterniones puramente imaginarios, $\mathrm{P_V} = \frac{\left(\mathrm{P} - \mathrm{P}^*\right)}{2} = 0 + iP_X + jP_Y + kP_Z$ y $\mathrm{Q_V} = \frac{\left(\mathrm{Q} - \mathrm{Q}^*\right)}{2} = 0 + iQ_X + jQ_Y + kQ_Z$ son paralelas; es decir, recordando la expresión 84 encontrada para el producto de dos quaterniones puramente imaginarios,

$$\mathrm{AB} = -\mathrm{A} \cdot \mathrm{B} + \mathrm{A} \times \mathrm{B} \tag{84}$$

si se hace $\mathrm{S} = \mathrm{P_V Q_V}$, entonces:

$$\mathrm{S} = \mathrm{P_V Q_V} = -\mathrm{P_V} \cdot \mathrm{Q_V} + \mathrm{P_V} \times \mathrm{Q_V} \tag{107}$$

Para saber si los quaterniones P y Q son paralelos, su parte vectorial $\mathrm{P_V} \times \mathrm{Q_V}$, debe ser igual a 0. De forma que:

$$\mathrm{S} - \mathrm{S}^* = 0 \tag{108}$$

De forma semejante, se dice que dos quaterniones P y Q son perpendiculares $\mathrm{P} \perp \mathrm{Q}$ si sus partes vectoriales $\mathrm{P_V}$ y $\mathrm{Q_V}$ son perpendiculares; es decir si $\mathrm{S} + \mathrm{S}^* = 0$. Dicho de otra forma, la parte escalar de S es igual a cero o el coseno del ángulo formado entre $\mathrm{P_V}$ y $\mathrm{Q_V}$ es cero.

5.2.3.10. Rotaciones con quaterniones

La aplicación más importante de los quaterniones es en la representación de orientaciones en el espacio; de la misma forma que los vectores en $\mathbb{R}^3$ de forma natural describen la posición en el espacio tridimensional, los quaterniones describen la orientación como una simple rotación en el espacio de cuatro dimensiones. A pesar de algunas opiniones encontradas [Gruber, 2000], [Grassia,

1998]; generalmente se considera más fácil utilizar quaterniones para representar rotaciones que cualquier otro método conocido [Horn, 1987], [Mukundan, 2002], [Hart, 1994], [Taylor, 1979], [Myung-Soo], [Pervin], [Shoemake, 2002], [Salamin, 1979], [Wisnesky, 2004]. Usar vectores en $\mathbb{R}^3$ para representar rotaciones es como si se quisiera usar coordenadas cilíndricas para representar traslaciones.

Dos tipos de rotaciones se pueden definir por medio de quaterniones; a saber, rotaciones con quaterniones puramente complejos y rotaciones con quaterniones unitarios. Y a su vez las rotaciones con quaterniones unitarios pueden ser divididas en rotaciones de quaterniones vectores perpendiculares al quaternión unitario y rotaciones de quaterniones vectores cualesquiera.

- ✓ Rotaciones con quaterniones puramente complejos.

Un quaternión puramente complejo se define como aquel que tiene su componente real y sólo uno de sus componentes imaginarios y el resto de sus componentes son igual a cero; por ejemplo tomando la componente imaginaria sobre el eje i.

$$Q_X = [w, x, 0, 0] \tag{109}$$

El álgebra de los quaterniones puramente complejos es isomorfa con el álgebra de los números complejos; por lo que se puede confiar en las propiedades de los números complejos para todas las operaciones de los quaterniones puramente complejos.

Sea el quaternión $U = [\cos\phi, \operatorname{sen}\phi, 0, 0]$. Claramente se nota que $U \in Q_X$ y que la norma del quaternión es igual a 1.

Sea también el quaternión $Q=[w,x,y,z]$ un quaternión cualquiera; el cual se puede expresar como $Q=Q_{||}+Q_{\perp}$, donde $Q_{||}=[w,x,0,0]\in Q_X$ y $Q_{\perp}=[0,0,y,z]\perp Q_X$.

Entonces $UQ=U(Q_{||}+Q_{\perp})=UQ_{||}+UQ_{\perp}$. Dado que $U,\ Q_{||}\in Q_X$, el primer término es simplemente una rotación de $Q_{||}$ un ángulo ϕ en el plano complejo wx.

Expandiendo el segundo término, siguiendo los elementos del quaternión producto definido en 78, se tiene $UQ_{\perp}=[0,0,y\cos\phi-z\,\mathrm{sen}\phi,z\cos\phi+y\,\mathrm{sen}\phi]$. Se ha encontrado entonces una rotación de $Q_{\perp}$ un ángulo ϕ en el plano yz.

De forma similar, teniendo $QU^*=(Q_{||}+Q_{\perp})U^*=Q_{||}U^*+Q_{\perp}U^*$. En este caso, $Q_{||}U^*\rightarrow qu^*=u^*q$ en el álgebra de complejos correspondiente, donde el complejo conjugado u^* representa una rotación de q un ángulo $-\phi$. Así $Q_{||}U^*$, representa una rotación de $Q_{||}$ un ángulo $-\phi$ en el plano wx. Sin embargo, expandiendo el segundo término se tiene que $Q_{\perp}U^*=[0,0,y\cos\phi-z\,\mathrm{sen}\phi,z\cos\phi+y\,\mathrm{sen}\phi]$. Se ha encontrado entonces también una rotación de $Q_{\perp}$ un ángulo ϕ en el plano yz.

Se pueden juntar estos productos $(UQ)U^*=UQU^*$, para definir un operador de rotación $\Re(U,Q)=UQU^*$, que rota cualquier quaternión Q un ángulo 2ϕ con respecto a U. En particular, los componentes y y z de Q se rotan con respecto a i en el plano yz, mientras que los componentes w y x quedan sin modificación. El vector $V=[0,x,y,z]$ es simplemente un subespacio de Q y se rota en a misma forma con respecto a U.

Nótese que a partir de la simetría en la definición de quaterniones, todo lo que se ha dicho hasta ahora para $\mathrm{U} \in \mathrm{Q}_\mathrm{X}$, aplica igualmente para $\mathrm{U} \in \mathrm{Q}_\mathrm{Y} = [\mathrm{w},0,\mathrm{y},0]$ y $\mathrm{U} \in \mathrm{Q}_\mathrm{Z} = [\mathrm{w},0,0,\mathrm{z}]$; que son entonces U_X, U_Y y U_Z, todos quaterniones unitarios puramente complejos. De forma que se puede decir

$$\begin{aligned} \Re(\mathrm{U}_\mathrm{X},\mathrm{Q}) &= \mathrm{U}_\mathrm{X}\mathrm{Q}\mathrm{U}_\mathrm{X}^*, \quad \mathrm{U}_\mathrm{X} = [\cos\phi,\mathrm{sen}\phi,0,0] \\ \Re(\mathrm{U}_\mathrm{Y},\mathrm{Q}) &= \mathrm{U}_\mathrm{Y}\mathrm{Q}\mathrm{U}_\mathrm{Y}^*, \quad \mathrm{U}_\mathrm{Y} = [\cos\phi,0,\mathrm{sen}\phi,0] \\ \Re(\mathrm{U}_\mathrm{Z},\mathrm{Q},) &= \mathrm{U}_\mathrm{Z}\mathrm{Q}\mathrm{U}_\mathrm{Z}^*, \quad \mathrm{U}_\mathrm{Z} = [\cos\phi,0,0,\mathrm{sen}\phi] \end{aligned} \tag{110}$$

Donde $\Re$ rota a Q un ángulo 2ϕ con respecto a U_K, donde $\mathrm{K} = \mathrm{X},\mathrm{Y},\mathrm{Z}$.

- ✓ Rotación de un quaternión vector por medio de un quaternión unitario perpendicular.

Sea $\mathrm{Q}_\mathrm{V} \in \mathrm{Im}\mathbb{H}$ un quaternión vector cualquiera y $\mathrm{U} = \cos\phi + U_V\,\mathrm{sen}\phi$ donde $U_V = [0,u_X,u_Y,u_Z]$; U es un quaternión unitario, tal que $\mathrm{Q}_\mathrm{V} \perp \mathrm{U}$. Multiplicando ambos quaterniones, se tiene:

$$T = \mathrm{U}\mathrm{Q}_\mathrm{V} = (\cos\phi + U_V\,\mathrm{sen}\phi)\mathrm{Q}_\mathrm{V} = \cos\phi\,\mathrm{Q}_\mathrm{V} + \mathrm{sen}\phi\,U_V\mathrm{Q}_\mathrm{V} \tag{111}$$

El primer término de la expresión del lado derecho de la igualdad es un quaternión vector $T_{V(1)} \| \mathrm{Q}_\mathrm{V}$. Dado que $\mathrm{Q}_\mathrm{V} \perp U_V$, el segundo término también debe ser un quaternión vector $T_{V(2)}$; y más aún, $T_{V(2)} \perp \mathrm{Q}_\mathrm{V}$ y $T_{V(2)} \perp \mathrm{U} \| U_V$. Y dado que T es el resultado de una suma de quaterniones vectores, debe ser también un quaternión vector. Tanto $T_{V(1)}$ como $T_{V(2)}$ se encuentran sobre un plano

perpendicular a U. De manera que $T = T_{V(1)} + T_{V(2)}$ se puede interpretar como una rotación de $\mathrm{Q_V}$ un ángulo ϕ con respecto a U en el plano perpendicular a U.

Ahora considerando el producto:

$$R = T\mathrm{U}^{-1} = T\mathrm{U}^{*} = T\cos\phi + TU_V^{*}\mathrm{sen}\phi = T\cos\phi - TU_V\mathrm{sen}\phi \tag{112}$$

Sabiendo que $TU_V = -U_V T$, la ecuación anterior se puede escribir como:

$$R = T\cos\phi + U_V T\mathrm{sen}\phi \tag{113}$$

Que es una rotación de T un ángulo ϕ con respecto a U. De las ecuaciones 109 y 110 se puede escribir entonces que:

$$R(\mathrm{U}, \mathrm{Q_V}) = \mathrm{UQ_V U}^{*} \tag{114}$$

Se puede decir que esta última expresión describe una rotación del quaternión vector $\mathrm{Q_V}$ un ángulo 2ϕ con respecto al quaternión unitario U, sabiendo que tales quaterniones son mutuamente perpendiculares.

- ✓ Rotación de un quaternión vector cualquiera por medio de un quaternión unitario.

Nótese la semejanza de la ecuación 112 con la expresión 108. Es de esperarse entonces que para describir una rotación de un quaternión cualquiera $\mathrm{Q} = [Q_W, Q_V]$ por un quaternión unitario U, la operación $R(\mathrm{U}, \mathrm{Q}) = \mathrm{UQU}^{*}$, llevará al quaternión $\mathrm{Q} = [Q_W, Q_V]$ a $\mathrm{Q'} = [Q_W, Q_V']$; cumpliendo

además que la norma del quaternión permanece inalterada $\|\mathrm{Q}\| = \|\mathrm{Q}'\|$. Para comprobar lo anterior, se considerará la fórmula para extraer la parte escalar de un quaternión,

$$2\mathrm{S}(\mathrm{Q}) = \mathrm{Q} + \mathrm{Q}^* \tag{115}$$

entonces para extraer la parte escalar de la expresión de rotación R, se aplica la ecuación 115,

$$2\mathrm{S}(R) = \mathrm{UQU}^* + \left(\mathrm{UQU}^*\right)^* = \mathrm{UQU}^* + \mathrm{UQ}^*\mathrm{U}^* \tag{116}$$

que factorizando, se tiene:

$$2\mathrm{S}(R) = \mathrm{U}\left(\mathrm{Q} + \mathrm{Q}^*\right)\mathrm{U}^* \tag{117}$$

Sustituyendo la ecuación 115 en la ecuación 117, se tiene que:

$$2\mathrm{S}(R) = \mathrm{U}\left(2\mathrm{S}(\mathrm{Q})\right)\mathrm{U}^* = 2\mathrm{S}(\mathrm{Q}) \tag{118}$$

Entonces la parte escalar del quaternión rotado es igual a la parte escalar del quaternión original.

La atención se enfoca ahora en la acción que realiza la operación $R(\mathrm{U},\mathrm{Q}) = \mathrm{UQU}^*$ sobre un quaternión vector cualquiera.

Considerando ahora que se tiene un quaternión vector arbitrario del tipo $\mathrm{Q_V} = [0, Q_X, Q_Y, Q_Z]$ y también un quaternión unitario $\mathrm{U} = \cos\phi + U_V \operatorname{sen}\phi$; entonces la operación $R(\mathrm{U},\mathrm{Q_V}) = \mathrm{UQ_VU}^*$ lleva al quaternión $\mathrm{Q_V}$ a una posición rotada un ángulo 2ϕ con respecto al eje U_V y como se demostró en la ecuación 116, la rotación no altera su norma.

5.2.3.11. Interpolación con quaterniones

Una de las aplicaciones más valiosas de los quaterniones es para calcular la interpolación de orientaciones de cuerpos en el espacio tridimensional; ésto sirve para hacer animaciones de los movimientos de los objetos en una escena al pasar de una orientación a otra, calculando una cantidad cualquiera de orientaciones intermedias, de forma que puedan ser proyectadas en la pantalla en forma secuencial dando la apariencia de movimiento suave.

Si se tienen dos orientaciones cualesquiera en el espacio, estas pueden ser definidas por dos quaterniones unitarios; llámense q_i y q_f, como quaternión inicial y quaternión final respectivamente. Existen infinidad de caminos a seguir para aplicar una rotación que vaya desde q_i hasta q_f; sin embargo, el camino más corto es aquel definido por una rotación directa simple, la cual forma una curva circular sobre la superficie de la hiperesfera unitaria S^3; esta curva circular cae donde la hiperesfera intersecta el plano formado por ambos quaterniones.

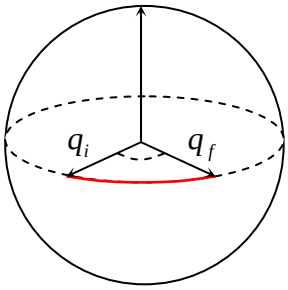


Figura 42. Rotación directa desde un quaternión q_i hasta q_f

Ken Shoemake [Shoemake, 1985] propuso dos fórmulas para realizar una interpolación lineal esférica con quaterniones, yendo desde q_i hasta q_f, con el parámetro u variando desde 0 hasta 1; la primera de ellas:

$$Slerp(q_i, q_f; u) = q_i \left(q_i^{-1} q_f\right)^u \tag{119}$$

y la otra, más práctica para programación:

$$Slerp(q_i, q_f; u) = \frac{sen\,(1-u)\theta}{sen\,\theta} q_i + \frac{sen\,(u\theta)}{sen\,\theta} q_f \tag{120}$$

5.3. Ejemplos de aplicación

5.3.1. Análisis de posición de un cuerpo rígido en el espacio

Como ejemplo de análisis de posición de un cuerpo rígido en el espacio se puede mostrar la rotación de un prisma cuadrático cuya sección transversal está formada por un cuadrado de lado unitario, y de longitud igual a tres; posicionado originalmente como se muestra en la Figura 43.

La posición de cualquier punto contenido dentro del cuerpo puede ser descrita por un vector en $\mathbb{R}^3$, y tal vector puede ser expresado a partir de las coordenadas desde dos diferentes bases; a saber, con referencia a una base inercial fija y también con referencia a una base móvil pegada al cuerpo mismo.

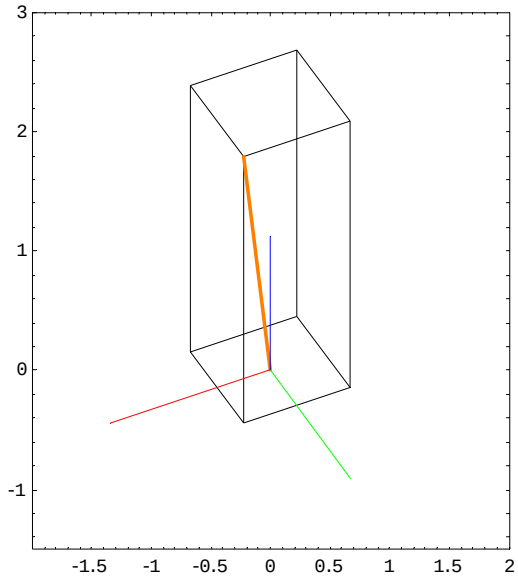


Figura 43. Prisma cuadrático posicionado en el origen de un sistema de coordenadas

Primero se define una base inercial fija de referencia, tal como sigue:

$$\mathrm{e}_i = \{\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3\} \in \mathbb{R}^3 \ : \ i = 1,2,3 \tag{121}$$

Donde $\mathrm{e}_1 = [1,0,0] = \hat{i}$, $\mathrm{e}_2 = [0,1,0] = \hat{j}$ y $\mathrm{e}_3 = [0,0,1] = \hat{k}$, conocidos como vectores canónicos.

Se define también una base móvil pegada al cuerpo en movimiento,

$$\mathrm{e}_i^1 = \{\mathrm{e}_1^1, \mathrm{e}_2^1, \mathrm{e}_3^1\} \in \mathbb{R}^3 \ : \ i = 1,2,3 \tag{122}$$

de modo que cada elemento de dicha base se expresa en referencia a la base inercial fija.

El vector de posición expresado con referencia a la base inercial es:

$$R_1 = [x_1, y_1, z_1]\mathrm{e}_i = x_1\hat{i} + y_1\hat{j} + z_1\hat{k} \tag{123}$$

donde x_1, y_1, z_1 son las coordenadas del vector de posición con respecto a la base inercial.

Debido a que la base móvil gira con respecto a la base inercial, a cada elemento de la base móvil se aplica la transformación definida en la ecuación 67, para expresarlos como elementos del espacio vectorial de los quaterniones, para después aplicar la rotación requerida por medio de la multiplicación por un quaternión unitario. Y por último, se aplica la transformación definida en la ecuación 66 para expresar el resultado como un elemento de $\mathbb{R}^3$. La base inercial se puede expresar entonces en términos de la base móvil de la siguiente forma:

$$\mathrm{e}_i = T_V\left(\mathrm{UT}_{\mathrm{Im}\mathbb{H}}\left(\mathrm{e}_i^1\right)\mathrm{U}^*\right) : i = 1,2,3 \tag{124}$$

Donde

$$\mathrm{U} = \left[\cos\left(\frac{\beta}{2}\right), \frac{x_R}{\|V_R\|}\operatorname{sen}\left(\frac{\beta}{2}\right), \frac{y_R}{\|V_R\|}\operatorname{sen}\left(\frac{\beta}{2}\right), \frac{z_R}{\|V_R\|}\operatorname{sen}\left(\frac{\beta}{2}\right)\right] \tag{125}$$

que es un quaternión unitario que especifica una rotación de un cuerpo rígido un ángulo (beta) β con respecto a un vector en $\mathbb{R}^3$, $V_R = [x_R, y_R, z_R]$.

La otra forma de expresar el vector de posición es con referencia a la base móvil:

$$R_1 = \left[x_1^1, y_1^1, z_1^1\right]\mathrm{e}_i^1 = x_1^1\,\hat{i}_1 + y_1^1\,\hat{j}_1 + z_1^1\,\hat{k}_1 \tag{126}$$

Donde x_1^1, y_1^1, z_1^1 son las coordenadas del vector de posición con respecto a la base móvil.

La base móvil se puede expresar en términos de la base inercial:

$$\mathrm{e}_i^1 = T_V\left(\mathrm{U}\mathrm{T}_{\mathrm{Im}\mathbb{H}}\left(\mathrm{e}_i\right)\mathrm{U}^*\right) : i = 1,2,3 \tag{127}$$

Donde

$$\mathrm{U} = \left[\cos\left(\frac{\beta}{2}\right), \frac{x_R}{\|V_R\|}\mathrm{sen}\left(\frac{\beta}{2}\right), \frac{y_R}{\|V_R\|}\mathrm{sen}\left(\frac{\beta}{2}\right), \frac{z_R}{\|V_R\|}\mathrm{sen}\left(\frac{\beta}{2}\right)\right] \tag{128}$$

que también es un quaternión unitario que especifica una rotación de un cuerpo rígido un ángulo (beta) β con respecto a un vector en $\mathbb{R}^3$, $V_R = \left[x_R, y_R, z_R\right]$.

Para dar una representación geométrica del cuerpo, se definen vectores en $\mathbb{R}^3$ que indiquen la posición con respecto a la base móvil; ésto es para cada uno de los vértices del prisma.

$$R_i^1 = \left[x_i^1, y_i^1, z_i^1\right] : i = 0,1,2,...,n \tag{129}$$

Donde n es la cantidad máxima de vectores requeridos para describir la geometría del cuerpo y $x_0^1 = 0.5$, $y_0^1 = 0.5$ y $z_0^1 = 3$ para el vector R_0^1 mostrado en naranja en la Figura 43, suponiendo que la base móvil se encuentra originalmente orientada en el mismo sentido de la base inercial.

Para efectuar una rotación sobre el cuerpo, a cada uno de los vectores que definen la posición de los vértices del cuerpo, se aplica el procedimiento que se describe a continuación:

Cada vector $R_i^1 : i = 0,1,2,...,n$ se expresa como elemento del espacio vectorial de los quaterniones aplicado la transformación definida en la ecuación 67, tal como sigue:

$$\mathrm{Q}_i^1 = T_{\mathrm{Im}\mathbb{H}}\left(R_i^1\right) = T_{\mathrm{Im}\mathbb{H}}\left(\left[x_i^1, y_i^1, z_i^1\right]\right) = \left[0, x_i^1, y_i^1, z_i^1\right] \tag{130}$$

Para realizar la rotación requerida, se aplica la transformación definida como rotación al quaternión anterior.

$$\mathrm{Q}_i' = R\left(\mathrm{U}, \mathrm{Q}_i^1\right) = \mathrm{U}\mathrm{Q}_i^1\mathrm{U}^* \tag{131}$$

Por último, se aplica la transformación definida en la ecuación 66 para representar un quaternión como elemento de $\mathbb{R}^3$ a cada uno de los quaterniones resultantes de la ecuación 127.

$$R_i = T_V\left(\mathrm{Q}_i'\right) = T_V\left(\mathrm{U}T_{\mathrm{Im}\mathbb{H}}\left(R_i^1\right)\mathrm{U}^*\right) \tag{132}$$

Para el prisma cuadrático del ejemplo, suponiendo que se quiere aplicar una rotación de 45º con respecto a un eje arbitrario orientado sobre el eje Z, el quaternión unitario que representa dicha rotación sería entonces, de la ecuación 124, se tiene:

$$\mathrm{U} = \left[\cos\left(\frac{45^\circ}{2}\right), 0, 0, \mathrm{sen}\left(\frac{45^\circ}{2}\right)\right] \tag{133}$$

Realizando las operaciones descritas en el proceso anterior, se encuentra que por ejemplo el vector de color naranja, $R_0 = \left[0, \sqrt{1/2}, 3\right]$. El prisma cuadrático del ejemplo rotado un ángulo de 45º con respecto al eje Z se presenta en la Figura 44.

5.3.2. Análisis de posición de dos cuerpos rígidos en el espacio

Como continuación del ejemplo anterior, se presenta el análisis de dos cuerpos rígidos en el espacio; siendo el primero de estos cuerpos un prisma cuadrático igual al del ejemplo anterior y el segundo

cuerpo otro prisma cuadrático de las mismas dimensiones posicionado encima del otro, con un ángulo de inclinación de -30º con respecto a la vertical; tal como se muestran en la Figura 45.

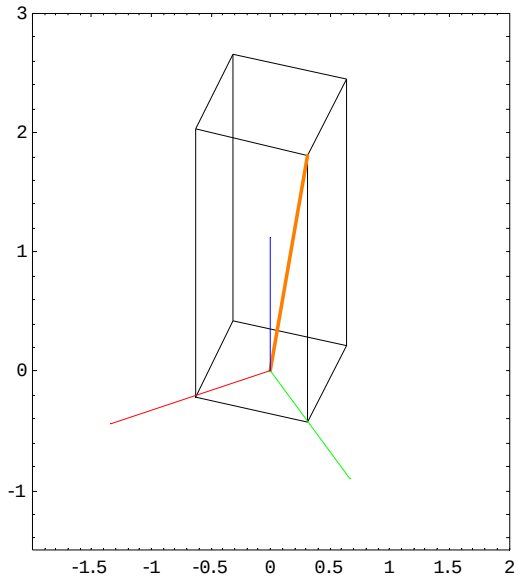


Figura 44. Prisma cuadrático girado 45º con respecto al eje Z

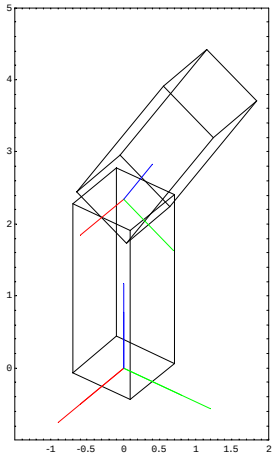


Figura 45. Dos prismas cuadráticos posicionados uno encima de otro

Primero, al igual que en el ejemplo anterior, se define una base inercial fija de referencia, tal como sigue:

$$\{e_1, e_2, e_3\} \in \mathbb{R}^3 \tag{134}$$

Donde $e_1 = [1,0,0]$, $e_2 = [0,1,0]$ y $e_3 = [0,0,1]$.

Se define también una base móvil pegada al primer cuerpo en movimiento, de la forma:

$$\{e_1^1, e_2^1, e_3^1\} \in \mathbb{R}^3 \tag{135}$$

De modo que cada elemento de dicha base se expresa en referencia a la base inercial fija.

Se define también una base móvil pegada al segundo cuerpo en movimiento, de la forma:

$$\{e_1^2, e_2^2, e_3^2\} \in \mathbb{R}^3 \tag{136}$$

Cuyos elementos también se expresan con respecto a la base inercial fija.

A continuación se definen vectores en $\mathbb{R}^3$ que especifican la posición de cada cuerpo con respecto a la base inercial. En la Figura 46 se muestran estos vectores.

Con base a los vectores declarados, se expresa una ecuación de posición que define la posición del extremo de la cadena cinemática. En la ecuación 135 se muestra esta ecuación:

$$R_P = R_1 + R_2 \tag{137}$$

Donde cada vector es expresado con respecto a la orientación de su base local.

El vector R_1 se expresa en función de la orientación de la base local del cuerpo 1, tal como sigue:

$$R_1 = r_1\, e_i^1 \ : \ i = 1,2,3 \tag{138}$$

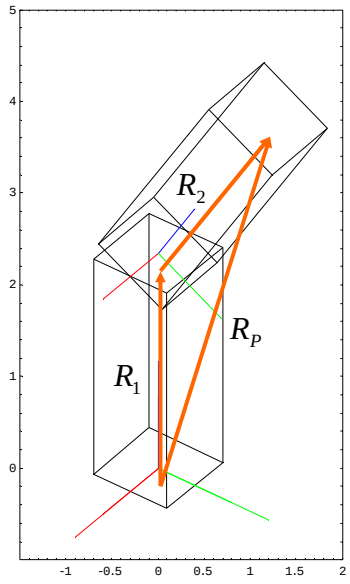


Figura 46. Representación de vectores de posición de dos cuerpos en el espacio

Donde r_1 es la longitud del cuerpo 1. La orientación de la base local 1 se expresa en función de la base inercial; de forma que:

$$R_1 = r_1 T_V\left(\mathrm{U}_1\ \mathrm{T}_{\mathrm{ImH}}(\mathrm{e}_i)\,\mathrm{U}_1^{*}\right) :\ i = 1,2,3 \tag{139}$$

Siendo U_1 un quaternión unitario que define la rotación del cuerpo 1, que tiene la forma:

$$\mathrm{U}_1 = \left[\cos\left(\frac{\theta_1}{2}\right), \frac{x_1}{\|V_1\|}\operatorname{sen}\left(\frac{\theta_1}{2}\right), \frac{y_1}{\|V_1\|}\operatorname{sen}\left(\frac{\theta_1}{2}\right), \frac{z_1}{\|V_1\|}\operatorname{sen}\left(\frac{\theta_1}{2}\right)\right] \tag{140}$$

Sabiendo que la rotación del cuerpo 1 es sobre el eje e_3, el quaternión unitario de rotación queda como sigue:

$$\mathrm{U}_1 = \left[\cos\left(\frac{\theta_1}{2}\right), 0, 0, \operatorname{sen}\left(\frac{\theta_1}{2}\right)\right] \tag{141}$$

El vector R_2 se expresa en función de la orientación de la base local del cuerpo 2, de la forma:

$$R_2 = r_2\, \mathrm{e}_i^2 \;:\; i = 1,2,3 \tag{142}$$

Donde r_2 es la longitud del cuerpo 2. La orientación de la base local 2 se expresa en función de la base local 1; de forma que:

$$R_2 = r_2\, T_V\left(\mathrm{U}_2{}'\, \mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_i^1\right)\mathrm{U}_2{}'^{*}\right) :\; i = 1,2,3 \tag{143}$$

Siendo U_2 un quaternión unitario que define la rotación del cuerpo 2, que tiene la forma:

$$\mathrm{U}_2 = \left[\cos\left(\frac{\theta_2}{2}\right), \frac{x_2}{\|V_2\|}\operatorname{sen}\left(\frac{\theta_2}{2}\right), \frac{y_2}{\|V_2\|}\operatorname{sen}\left(\frac{\theta_2}{2}\right), \frac{z_2}{\|V_2\|}\operatorname{sen}\left(\frac{\theta_2}{2}\right)\right] \tag{144}$$

Si se quiere que la rotación del cuerpo 2 sea sobre el eje e_1^1, el quaternión unitario de rotación queda como sigue:

$$\mathrm{U}_2 = \left[\cos\left(\frac{\theta_2}{2}\right), \operatorname{sen}\left(\frac{\theta_2}{2}\right), 0, 0\right] \tag{145}$$

Sin embargo; Este quaternión U_2, es afectado por la rotación definida por el quaternión U_1. De forma que $\mathrm{U}_2{}' = \mathrm{U}_1\, \mathrm{U}_2\, \mathrm{U}_1^{*}$. Sustituyendo las ecuaciones 135 y 139 en la ecuación 133, la ecuación de posición queda:

$$R_P = r_1\, T_V\left(\mathrm{U}_1\, \mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_i\right)\mathrm{U}_1^{*}\right) + r_2\, T_V\left(\mathrm{U}_2\, \mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_i^1\right)\mathrm{U}_2{}^{*}\right) \tag{146}$$

Para dar una representación geométrica del cuerpo 1; de igual forma como se hizo para el ejemplo anterior, se definen vectores en $\mathbb{R}^3$ que indiquen la posición con respecto a la base móvil 1; ésto es para cada uno de los vértices del prisma,

$$R_i^1 = \left[x_i^1, y_i^1, z_i^1\right] : \ i = 0,1,2,...,n \tag{147}$$

Donde n es la cantidad máxima de vectores requeridos para describir la geometría del cuerpo y x_i^1, y_i^1 y z_i^1 son las componentes del i-ésimo vector.

Para efectuar una rotación sobre el cuerpo, a cada uno de los vectores $R_i^1 \ : \ i = 0,1,2,...,n$ que definen la posición de los vértices del cuerpo, se aplica la transformación siguiente:

$$R_i = T_V\left(\mathrm{U}_1 T_{\mathrm{Im}\mathbb{H}}\left(R_i^1\right)\mathrm{U}_1^*\right) \tag{148}$$

Para dar la representación geométrica del cuerpo 2; de forma semejante a como se hizo para el cuerpo 1, se definen vectores en $\mathbb{R}^3$ que indiquen la posición con respecto a la base móvil 2; ésto es para cada uno de los vértices del prisma,

$$R_j^2 = \left[x_j^2, y_j^2, z_j^2\right] : \ j = 0,1,2,...,n_2 \tag{149}$$

Donde n_2 es la cantidad máxima de vectores requeridos para describir la geometría del cuerpo 2 y x_j^2, y_j^2 y z_j^2 son las componentes del j-ésimo vector.

Para efectuar una rotación sobre el cuerpo, a cada uno de los vectores $R_j^2 \ : \ j = 0,1,2,...,n_2$ que definen la posición de los vértices del cuerpo 2, se aplica la transformación siguiente:

$$R_j = R_1 + T_V\left(\mathrm{U}_1\mathrm{U}_2{}'T_{\mathrm{ImH}}\left(R_j^2\right)\mathrm{U}_2{}'^{*}\,\mathrm{U}_1^{*}\right) \tag{150}$$

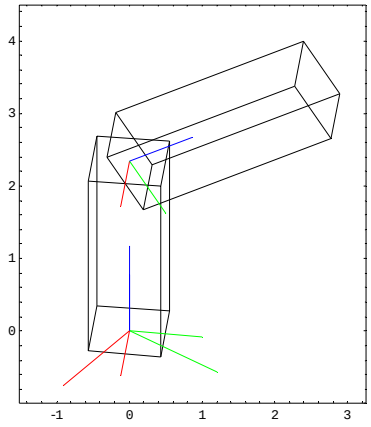


Figura 47. Dos cuerpos en el espacio girados con respecto a su posición original

6. Modelo matemático de la cinemática del Robot CRS A465

6.1. Arquitectura del Robot CRS A465

El Robot CRS A465 es un brazo articulado de 6 grados de libertad con juntas rotacionales en todas sus articulaciones, construido bajo una configuración de cadena cinemática abierta. Ver Figura 48. La posición general del órgano efector está dada por la posición de las tres primeras articulaciones de la cadena $\theta_1, \theta_2,$ y θ_3.

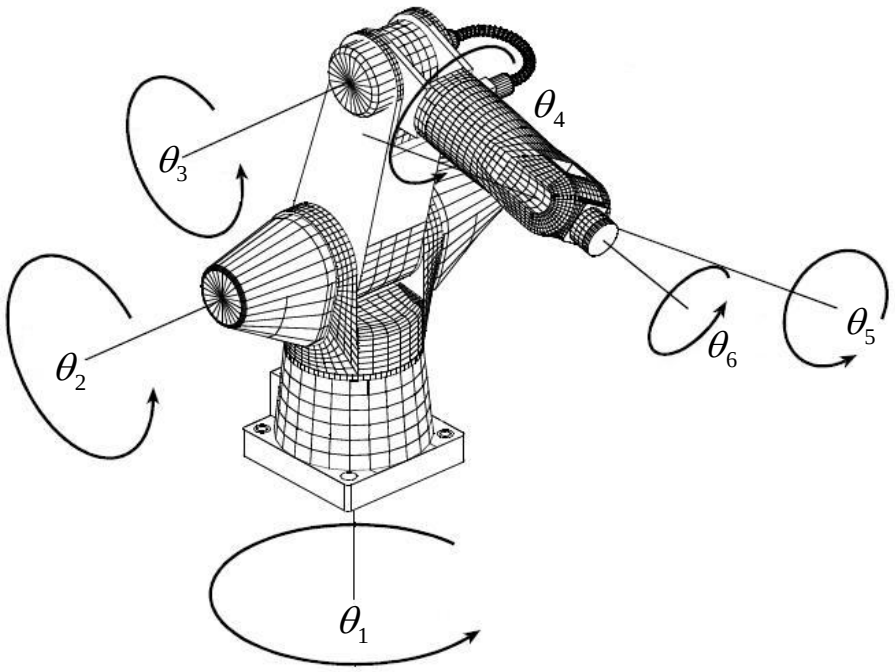


Figura 48. Articulaciones del Robot CRS A465

La orientación del órgano efector esta dada a su vez por las últimas tres articulaciones de la cadena θ_4, θ_5, y θ_6, a las cuales se les conoce como Alabeo de muñeca (Yaw), Cabeceo de muñeca (Pitch) y Rotación de muñeca (Roll).

Las dimensiones de los eslabones y el rango de movimiento de cada uno de ellos describiendo el área de trabajo del robot, se presentan en la Figura 49 y en la Figura 50.

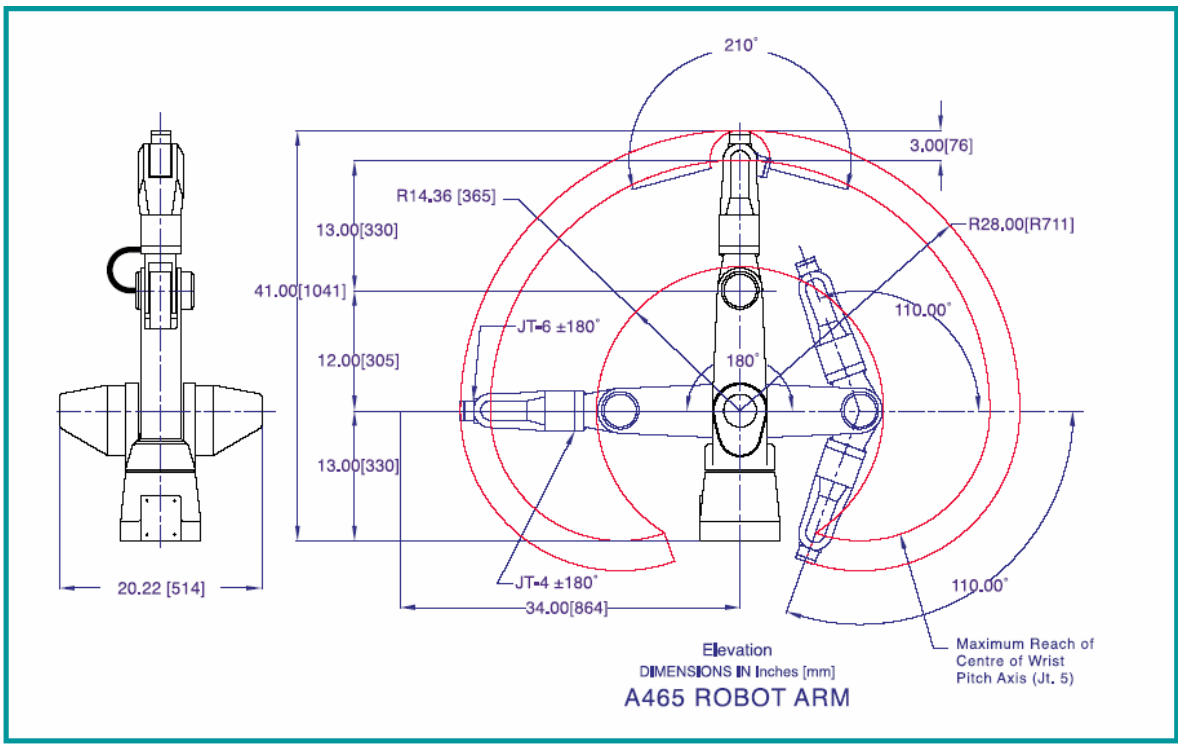


Figura 49. Área de trabajo del Robot CRS A465 (vista lateral)

6.2. Análisis de posición del robot CRS A465 (Cinemática Directa)

Para lograr describir la cinemática del robot CRS A465, primero se construye el modelo matemático que ayude a analizar la posición tanto de los eslabones como del órgano efector. A esto se le conoce

como problema cinemático directo; es decir, encontrar la posición y orientación del órgano efector del robot a partir de los valores de los ángulos de rotación de las articulaciones. Para construir el modelo matemático, primero se define una base inercial u origen de referencia para el modelo matemático que describirá los movimientos del brazo en un espacio tridimensional. Dicha base inercial se considera fija o de referencia y estará descrita en términos de vectores ortonormales en $\mathbb{R}^3$.

$$\begin{aligned} e_1 &= [1,0,0] = \hat{i} \\ e_2 &= [0,1,0] = \hat{j} \\ e_3 &= [0,0,1] = \hat{k} \end{aligned} \tag{151}$$

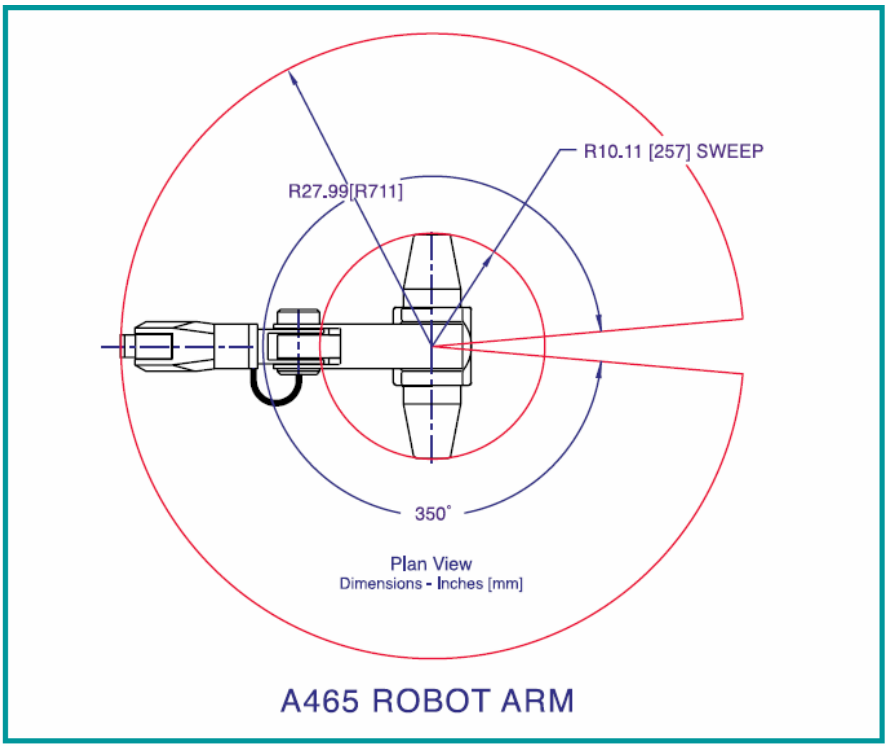


Figura 50. Área de trabajo del Robot CRS A465 (vista superior)

Se definen ahora vectores en $\mathbb{R}^3$ para especificar la posición de cada uno de los eslabones del robot. En la Figura 51 se muestran los vectores declarados; nótese que en la base del robot se

establece un marco de referencia tridimensional XYZ coincidente con la base inercial declarada en 119, donde los vectores unitarios $\hat{i}$, $\hat{j}$ y $\hat{k}$ se representan por flechas de color rojo, verde y azul respectivamente. Existe un vector denotado por R_0 que establece la posición del centro de rotación θ_1 (cintura) del cuerpo 2 (tronco); otro vector denotado por R_1 que establece la posición con respecto al extremo del vector anterior R_0, del centro de rotación θ_2 (hombro) del cuerpo 3 (brazo); un tercer vector R_2 que establece la posición con respecto al extremo del vector anterior R_1, del centro de rotación θ_3 (codo) del cuerpo 4 (antebrazo); un cuarto vector R_3 que establece a su vez la posición con respecto al extremo del vector anterior R_2, del centro de rotación θ_4 (muñeca) del cuerpo 5 (mano), un quinto vector R_4, que describe la posición de la mano o del órgano efector con respecto al extremo del vector anterior R_3; y finalmente un vector R_P que describe la posición de la mano o del órgano efector con respecto al origen del marco de referencia XYZ de la base inercial.

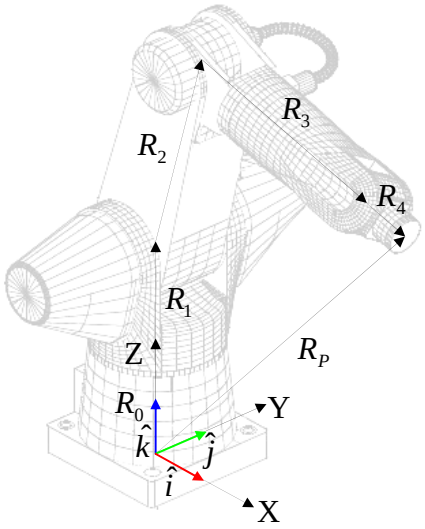


Figura 51. Representación con vectores del Robot CRS A465

Ahora; como se muestra en la Figura 52, se declaran bases locales en cada uno de los centros de rotación; usando bases del tipo dextrógiro solamente y con el vector $\hat{k}$ correspondiente, siempre apuntando hacia el extremo del robot; tal como se muestra.

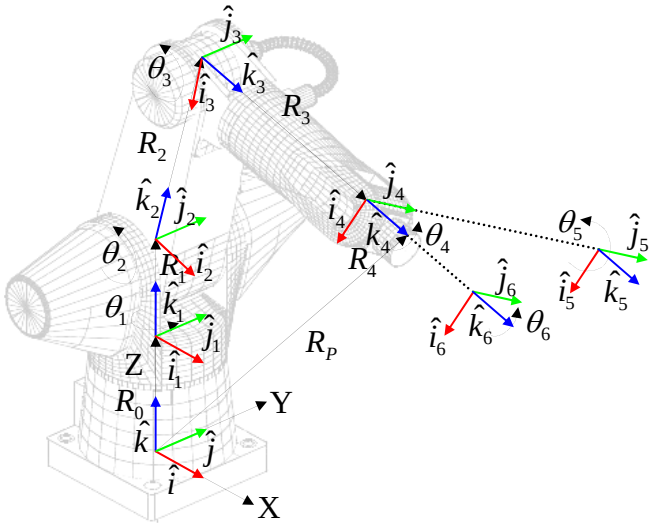


Figura 52. Declaración de bases locales del Robot CRS A465

Nótese que se hace coincidir uno de los vectores canónicos de cada base móvil con el eje de rotación de cada articulación

Se puede determinar una ecuación de posición

$$R_P = R_0 + R_1 + R_2 + R_3 + R_4 \tag{152}$$

donde R_0 se asocia con la dimensión r_0 y a la base inercial; y sabiendo que el vector se encuentra orientado en el sentido positivo del eje $\hat{k}$, entonces,

$$R_0 = r_0\hat{k} = [0,0,r_0] \tag{153}$$

La posición del segundo eslabón o tronco del robot, se asocia a la magnitud del vector R_1 y a la orientación de su base móvil; tal como sigue:

$$R_1 = r_1 \mathrm{e}_i^1 \ : \ i = 1,2,3 \tag{154}$$

y la orientación de la base móvil se expresa en función de la base inercial de la siguiente manera:

$$R_1 = r_1 \, T_V \left(\mathrm{U}_1 \, \mathrm{T}_{\mathrm{ImH}} \left(\mathrm{e}_i \right) \mathrm{U}_1^{\ *} \right) : \ i = 1,2,3 \tag{155}$$

Siendo U_1 un quaternión unitario que define la rotación del cuerpo 1, que tiene la forma:

$$\mathrm{U}_1 = \left[\cos\left(\frac{\theta_1}{2} \right), \frac{x_1}{\|V_1\|} \operatorname{sen}\left(\frac{\theta_1}{2} \right), \frac{y_1}{\|V_1\|} \operatorname{sen}\left(\frac{\theta_1}{2} \right), \frac{z_1}{\|V_1\|} \operatorname{sen}\left(\frac{\theta_1}{2} \right) \right] \tag{156}$$

Sabiendo que la rotación del tronco es sobre el eje $\hat{k}_1$, el quaternión unitario de rotación queda como sigue:

$$\mathrm{U}_1 = \left[\cos\left(\frac{\theta_1}{2} \right), 0, 0, \operatorname{sen}\left(\frac{\theta_1}{2} \right) \right] \tag{157}$$

La posición del tercer eslabón o brazo del robot, se asocia a la magnitud del vector R_2 y a la orientación de su base móvil; tal como sigue:

$$R_2 = r_2 \mathrm{e}_i^2 \ : \ i = 1,2,3 \tag{158}$$

y la orientación de la base móvil e_i^2 se expresa en función de la base móvil anterior e_i^1 de la siguiente manera:

$$R_2 = r_2\, T_V\left(\mathrm{U}_2'\,\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_{\mathrm{i}}^{1}\right)\mathrm{U}_2'^{*}\right) :\ i = 1,2,3 \tag{159}$$

Siendo U_2 un quaternión unitario que define la rotación del brazo; y sabiendo que la rotación del brazo es sobre el sentido negativo del eje $\hat{j}_2$, tiene la forma:

$$\mathrm{U}_2 = \left[\cos\left(\frac{\theta_2}{2}\right),0,-\mathrm{sen}\left(\frac{\theta_2}{2}\right),0\right] \tag{160}$$

Hasta el momento, este planteamiento coincide con el planteamiento hecho por Márquez [Márquez, 2000] Sin embargo; el quaternión U_2 define una rotación sobre el eje negativo de $\hat{j}_2$, pero este eje es afectado por la rotación aplicada por U_1. Por lo que la rotación definida por U_2, debe ser afectada también por la rotación de U_1. Entonces, el quaternión que define la rotación del vector R_2, es

$$\mathrm{U}_2' = \mathrm{U}_1\,\mathrm{U}_2\,\mathrm{U}_1^{*} \tag{161}$$

Lo cual es diferente al planteamiento hecho por Márquez en su tesis.

La posición del cuarto eslabón o antebrazo del robot, se asocia a la magnitud del vector R_3 y a la orientación de su base móvil; tal como sigue:

$$R_3 = r_3\mathrm{e}_i^3\ :\ i = 1,2,3 \tag{162}$$

y la orientación de la base móvil e_i^3 se expresa en función de la base móvil anterior e_i^2 de la siguiente manera:

$$R_3 = r_3\, T_V\left(\mathrm{U_3'\, T_{ImH}}\left(\mathrm{e_i^2}\right)\mathrm{U_3'^{*}}\right) :\ i = 1,2,3 \tag{163}$$

Siendo $\mathrm{U_3}$ un quaternión unitario que define la rotación del antebrazo; y sabiendo que la rotación de este cuerpo es sobre el sentido negativo del eje $\hat{j}_3$, tiene la forma:

$$\mathrm{U_3} = \left[\cos\left(\frac{\theta_3}{2}\right), 0, -\mathrm{sen}\left(\frac{\theta_3}{2}\right), 0\right] \tag{164}$$

Sin embargo; el quaternión $\mathrm{U_3}$ define una rotación sobre el eje negativo de $\hat{j}_3$, pero este eje, en forma semejante al quaternión $\mathrm{U_2}$ es afectado por las rotaciones aplicadas tanto por $\mathrm{U_1}$ como por $\mathrm{U_2}$. Por lo que la rotación definida por $\mathrm{U_3}$, debe ser afectada también por las rotaciones de $\mathrm{U_1}$ y $\mathrm{U_2}$. Entonces, el quaternión que define la rotación del vector R_3, es

$$\mathrm{U_3' = U_1\, U_2 U_3\, U_2^{*} U_1^{*}} \tag{165}$$

La posición y la orientación del último eslabón o mano del robot, se asocia a la magnitud del vector R_4 y a la combinación de orientaciones conocidas como alabeo (yaw), cabeceo (pitch) y giro (roll) del órgano efector; que con respecto a la base inercial, es la orientación de la última base móvil definida para el robot. Tal como se muestra a continuación:

$$R_4 = r_4 \mathrm{e}_i^6\ :\ i = 1,2,3 \tag{166}$$

y la orientación de la base móvil e_i^6 se expresa en función de la base móvil del antebrazo e_i^3 de la siguiente manera:

$$R_4 = r_4\, T_V\left(\mathrm{U}_6'\left(\mathrm{U}_5'\left(\mathrm{U}_4'\left(\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_{\mathrm{i}}^3\right)\right)\mathrm{U}_4'^*\right)\mathrm{U}_5'^*\right)\mathrm{U}_6'^*\right) :\ i = 1,2,3 \tag{167}$$

Los quaterniones unitarios que definen esta orientación son:

$$\mathrm{U}_4 = \left[\cos\left(\frac{\theta_4}{2}\right), 0, 0, \mathrm{sen}\left(\frac{\theta_4}{2}\right)\right] \tag{168}$$

$$\mathrm{U}_5 = \left[\cos\left(\frac{\theta_5}{2}\right), 0, -\mathrm{sen}\left(\frac{\theta_5}{2}\right), 0\right] \tag{169}$$

$$\mathrm{U}_6 = \left[\cos\left(\frac{\theta_6}{2}\right), 0, 0, \mathrm{sen}\left(\frac{\theta_6}{2}\right)\right] \tag{170}$$

De forma semejante a lo visto anteriormente con los vectores R_2 y R_3, los quaterniones U_4, U_5 y U_6 son afectados escalonadamente por las rotaciones definidas por los quaterniones U_1, U_2 y U_3 y entre ellos mismos; de forma que,

$$\mathrm{U}_4' = \mathrm{U}_1\,\mathrm{U}_2\mathrm{U}_3\,\mathrm{U}_4\mathrm{U}_3^*\mathrm{U}_2^*\mathrm{U}_1^* \tag{171}$$

$$\mathrm{U}_5' = \mathrm{U}_1\,\mathrm{U}_2\mathrm{U}_3\,\mathrm{U}_4\mathrm{U}_5\mathrm{U}_4^*\mathrm{U}_3^*\mathrm{U}_2^*\mathrm{U}_1^* \tag{172}$$

$$\mathrm{U}_6' = \mathrm{U}_1\,\mathrm{U}_2\mathrm{U}_3\,\mathrm{U}_4\mathrm{U}_5\mathrm{U}_6\mathrm{U}_5^*\mathrm{U}_4^*\mathrm{U}_3^*\mathrm{U}_2^*\mathrm{U}_1^* \tag{173}$$

Sustituyendo las ecuaciones 153, 155, 159, 163 y 167 en la ecuación de posición 152, la posición del órgano efector con referencia a la base inercial, queda definida como sigue:

$$\begin{aligned} R_P = {} & r_0\hat{k} + r_1\, T_V\left(\mathrm{U}_1\,\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_i\right)\mathrm{U}_1^*\right) + r_2\, T_V\left(\mathrm{U}_2'\,\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_{\mathrm{i}}^1\right)\mathrm{U}_2'^*\right) \\ & + r_3\, T_V\left(\mathrm{U}_3'\,\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_{\mathrm{i}}^2\right)\mathrm{U}_3'^*\right) + r_4\, T_V\left(\mathrm{U}_6'\left(\mathrm{U}_5'\left(\mathrm{U}_4'\left(\mathrm{T}_{\mathrm{ImH}}\left(\mathrm{e}_{\mathrm{i}}^3\right)\right)\mathrm{U}_4'^*\right)\mathrm{U}_5'^*\right)\mathrm{U}_6'^*\right) \end{aligned} \tag{174}$$

Que expresada en términos de ángulos de rotación, queda como sigue:

$$
\begin{aligned}
R_P = \big[& r_4 \sin(\theta_1)\sin(\theta_4)\sin(\theta_5) - \cos(\theta_1)(r_2 \sin(\theta_2) + (r_3 + r_4 \cos(\theta_5))\sin(\theta_2 + \theta_3) + \\
& r_4 \cos(\theta_2 + \theta_3)\cos(\theta_4)\sin(\theta_5)), -r_4 \cos(\theta_1)\sin(\theta_4)\sin(\theta_5) - \sin(\theta_1)(r_2 \sin(\theta_2) + \\
& (r_3 + r_4 \cos(\theta_5))\sin(\theta_2 + \theta_3) + r_4 \cos(\theta_2 + \theta_3)\cos(\theta_4)\sin(\theta_5)), r_0 + r_1 + \\
& r_3 \cos(\theta_2 + \theta_3) + \cos(\theta_2)(r_2 + r_4 \cos(\theta_3)\cos(\theta_5)) - r_4(\cos(\theta_5)\sin(\theta_2)\sin(\theta_3) + \\
& \cos(\theta_4)\sin(\theta_2 + \theta_3)\sin(\theta_5))\big]
\end{aligned}
\tag{175}
$$

6.3. Cinemática inversa

El problema cinemático inverso, tal como su nombre lo indica, es lo contrario al problema cinemático directo; es decir, a partir de una posición y orientación específicas del órgano efector del robot, encontrar los valores de los ángulos de rotación de las articulaciones del robot.

Hasta el momento existen algunos métodos para dar solución a la cinemática inversa; entre ellos se puede mencionar el método puramente geométrico, como el descrito en [Esteve, 2001] y en [Universidad de Vigo], otro método, pero del tipo numérico muy conocido es utilizando una variante del Método numérico Newton-Raphson para solución de ecuaciones simultáneas no lineales. En [Welman, 1993], Chris Welman presenta una comparación de dos métodos para solución de la cinemática inversa; uno de ellos utilizando la transpuesta de una matriz Jacobiana y el otro llamado Método descendiente cíclico de coordenadas, CCD por sus siglas en inglés (Ciclic-Coordinate Descendent). En esta comparación Welman presenta unas gráficas diciendo que el método CCD ocupa una cuarta parte del tiempo de convergencia comparado con el otro método.

6.3.1. Método descendente cíclico de coordenadas

El método CCD es un método cíclico heurístico que intenta minimizar el error de posición y orientación al hacer variar una articulación a la vez. En cada ciclo se hace variar una articulación a la vez comenzando desde la articulación más cercana al órgano efector hasta la articulación más cercana a la base del manipulador. Conforme se va obteniendo un resultado parcial para cada articulación, este resultado se aplica al manipulador de forma que las siguientes articulaciones a ser analizadas tomen en cuenta este nuevo resultado.

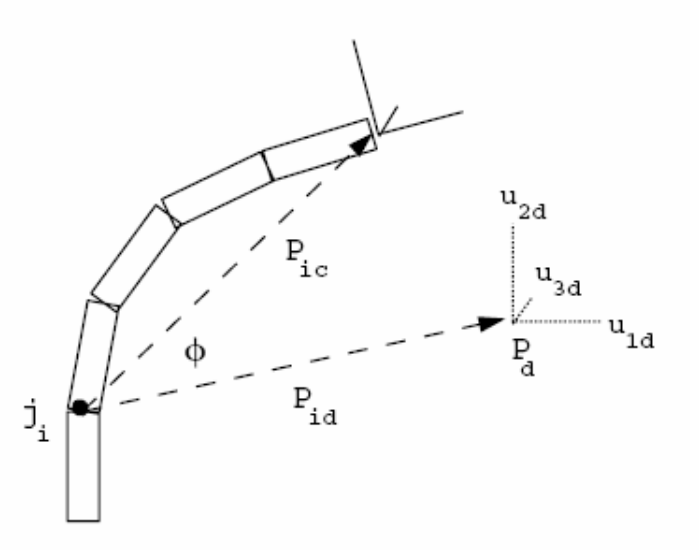


Figura 53. Ejemplo de aplicación del método CCD para la articulación i

Suponiendo que la posición actual de un brazo manipulador es el punto $P_c = [x_c, y_c, z_c]$, y la orientación está dada por los tres vectores ortonormales $O_c = \begin{bmatrix} u_{1c} \\ u_{2c} \\ u_{3c} \end{bmatrix}$. Y la posición deseada es el

punto $P_d = [x_d, y_d, z_d]$, con la orientación deseada $O_d = \begin{bmatrix} u_{1d} \\ u_{2d} \\ u_{3d} \end{bmatrix}$, el órgano efector se puede poner en una posición y orientación tan cerca como sea posible del punto deseado al encontrar un valor de rotación ϕ para la articulación en turno j_i, que ayude a minimizar la siguiente expresión

$$E(q) = E_p(q) + E_o(q) \tag{176}$$

Que no es más que la suma del error de posición

$$E_p(q) = \|(P_d - P_c)\| \tag{177}$$

y el error de orientación

$$E_o(q) = \sum_{j=1}^{3} \left((u_{jd} \cdot u_{jc}) - 1 \right)^2 \tag{178}$$

Con referencia a la Figura 53, se presenta un ejemplo de una articulación j_i rotacional a partir de la cual se puede trazar una línea recta hacia la posición actual del órgano efector y otra línea recta hacia la posición deseada del órgano efector. Se puede hacer girar la articulación j_i con respecto a un eje perpendicular al plano formado por las dos líneas; conforme varía el ángulo de rotación ϕ, se forma un círculo con centro en j_i y radio igual a la distancia entre j_i y el punto P_c. El punto sobre este círculo más cercano al punto de posición deseado es aquel donde el círculo intersecta con la línea entre j_i y el punto deseado. Si se traza un vector de posición desde j_i a P_c, se le puede

llamar P_{ic}, y otro vector desde j_i a P_d, llamado entonces P_{id}; el vector P_{ic} rotado con respecto a su eje un ángulo ϕ, sería

$$P_{ic}'(\phi) = R_i(\phi)P_{ic} \tag{179}$$

Lo que se busca entonces es alinear los dos vectores $P'_{ic}(\phi)$ y P_{id}; que dicho de otra forma, se busca un valor de rotación ϕ que ayude a maximizar la expresión

$$g_p(\phi) = P_{id} \cdot P'_{ic}(\phi) \tag{180}$$

y pensando en forma similar, el error de orientación se reduce cuando el ángulo ϕ también maximiza la expresión

$$g_o(\phi) = \sum_{j=1}^{3} u_{jd} \cdot u'_{jc}(\phi) \tag{181}$$

Welman en su trabajo [Welman, 1993] presenta una combinación de estas dos últimas expresiones, pero agrega dos factores de ponderación; tal como sigue

$$g(\phi) = w_p g_p(\phi) + w_o g_o(\phi) \tag{182}$$

Estos factores de ponderación, son valores arbitrarios que se proponen como $w_o = 1$ para el factor de ponderación de orientación y $0 \le w_p \le 1$ para el factor de ponderación de la posición. Estos factores afectan el comportamiento de rigidez de la articulación en juego; característica aprovechada por Johnson [Johnson, 2003] para dar expresividad a los movimientos de un cuerpo articulado; como él mismo lo expresa en su trabajo, suponiendo que se tiene un perro como personaje de una escena,

la cinemática inversa normal tendría que resolver el problema “quiero poner mi pata en ese punto”; sin embargo, este tipo de algoritmo daría una apariencia “robótica” al personaje y lo que se pretende con la cinemática inversa expresiva es resolver el problema “quiero poner mi pata en ese punto, pero estoy muy cansado”.

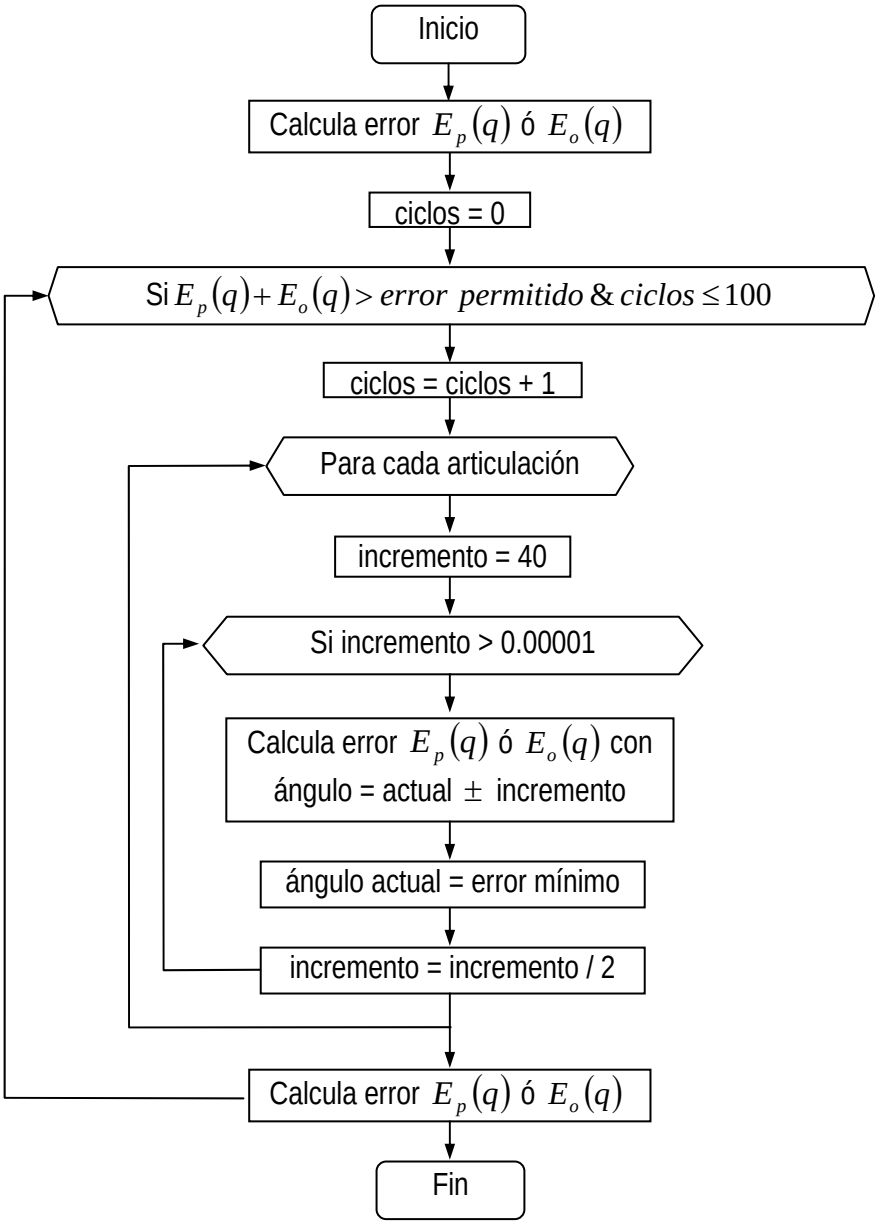


Figura 54. Diagrama de flujo del algoritmo CCD

7. Programa para simulación de la cinemática del robot CRS A465

El programa de simulación desarrollado como resultado de este trabajo de tesis, ha sido pensado con el fin de ofrecer al alumnado, estudiantes de alguna materia relacionada con el manejo del robot CRS A465 del ITESCA, una alternativa para el aprendizaje del manejo de dicho robot; desde cualquier computadora personal disponible. Presentando al usuario una interfase gráfica interactiva con una imagen del robot construido con primitivas sólidas sombreadas para mayor realismo.

El usuario tendrá la capacidad de cambiar el punto de vista para ver la escena del robot desde cualquier ángulo deseado; así también tendrá la capacidad de regresar el punto de vista al punto original. Se proporciona también una emulación de un teclado como el del control manual del robot, llamado “Teach pendant” en inglés. Con este teclado, el usuario será capaz de interactuar con el robot para efectuar movimientos y grabar posiciones. Junto a la emulación del teclado, se presenta también una pantalla semejante a la original del teach pendant, donde se presenta al usuario los mensajes correspondientes a la operación realizada. Por último, se presenta una sección con el resultado de posición del órgano efector obtenido con los cálculos de la cinemática directa del robot.

El desarrollo del programa fue realizado completamente con el lenguaje de programación Visual Basic V6.0, bajo el ambiente de Windows®; utilizando para el manejo de gráficos, unas librerías tipo OpenGL disponibles para Visual Basic.

7.1. Interfase para el usuario

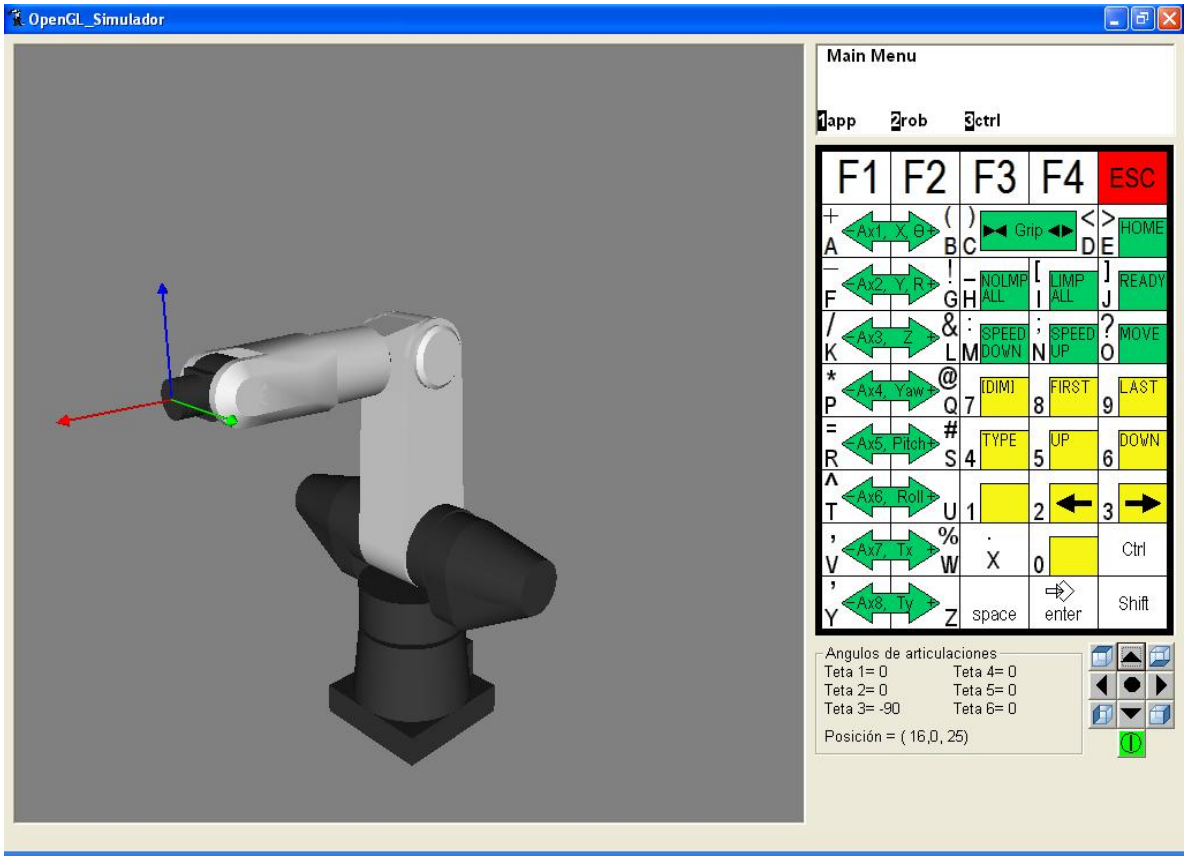


Figura 55. Pantalla principal del simulador para el robot CRS A465

El programa cuenta con una sola pantalla interfase para el usuario; tal como se muestra en la Figura 55. En ella se puede ver a la izquierda una imagen sombreada representativa del robot CRS A465, visto desde un punto de vista con una posición equidistante con respecto a un marco de referencia tridimensional; cuyo origen se encuentra en el centro del plano inferior de la base del robot, con el eje X orientado hacia el frente del robot; es decir hacia donde se encuentra apuntando el antebrazo del mismo. El eje Y se encuentra orientado en forma perpendicular al eje X apuntando hacia la derecha de la pantalla sobre el mismo plano inferior de la base del robot. Y por último, el eje Z se

encuentra orientado de manera que forma un sistema de referencia del tipo dextrógiro. Dicho sistema de referencia, es el mismo definido como base inercial mostrado en la Figura 51. Después de iniciar el programa será necesario activar la imagen del robot; para hacerlo, es necesario presionar el botón de encendido  .

En la parte superior derecha de la interfase, se encuentra la pantalla del teach pendant; donde se presentan los mensajes escritos presentados al usuario.

Bajo la pantalla anterior, se presenta la emulación del teclado del teach pendant.

Más abajo, se presenta el resultado del cálculo de la cinemática directa del robot, mostrando el valor de cada uno de los ángulos de rotación de cada articulación; así como el resultado de la posición del órgano efector con respecto al sistema de referencia global o inercial.

En la parte inferior derecha de la interfase, se presenta un conjunto de cinco botones que sirven para cambiar la ubicación del punto de vista.

7.2. Punto de vista

Como se ha venido diciendo, el usuario puede en cualquier momento, cambiar la ubicación del punto de vista; de manera que la imagen pueda ser apreciada desde un punto arbitrario, tal como un camarógrafo puede cambiar la ubicación de su cámara para captar diferente toma de una escena. En la Figura 56 se muestra un ejemplo de la vista superior e inferior del robot.

7.2.1. Cómo mover el punto de vista

El usuario puede mover la ubicación del punto de vista, al hacer “clic” con el apuntador sobre cualquiera de los cuatro botones con flecha del arreglo de botones presentado para este propósito en la parte inferior derecha de la interfase. Los botones con flecha hacia arriba  y hacia abajo  provocarán un giro de 5º hacia arriba o hacia abajo, según corresponda del punto de vista, tomando como eje de rotación la línea central horizontal de la imagen. A su vez, los botones con flecha hacia la derecha  o a la izquierda  provocarán un giro de 5º en sentido positivo o negativo con respecto al eje Z del marco de referencia inercial; o bien, la vertical del robot.

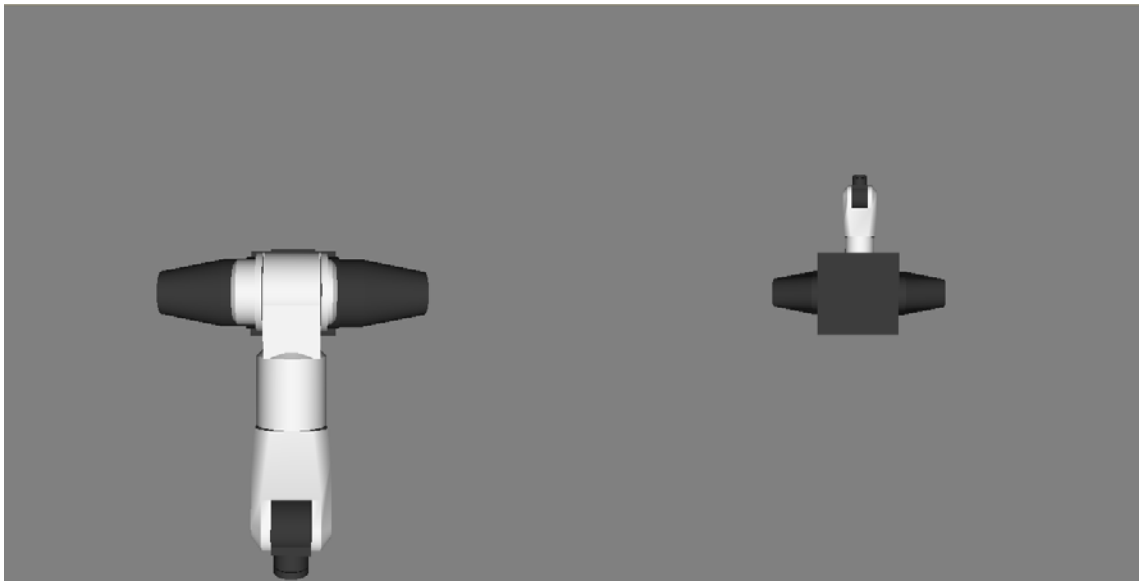


Figura 56. Vista superior e inferior del robot CRS A465

Se ofrecen también cuatro botones más para mover el punto de vista hacia cuatro posiciones predeterminadas; a saber, éstos dan una vista superior  , vista inferior  , vista lateral a la

izquierda  y vista lateral a la derecha  de la escena. Al presionar cualquier botón de los anteriores, el programa calculará la ruta a seguir para lograr orientar el punto de vista a su origen a partir del punto de vista actual y por medio de quaterniones hará una interpolación para encontrar 200 orientaciones intermedias para lograr una transición de la escena en forma suavizada hasta llegar a la orientación del punto de vista deseado

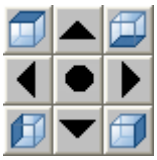


Figura 57. Botones de control para mover el punto de vista

7.2.2. Cómo regresar al punto de vista original

Para regresar el punto de vista a su posición de origen, tal como cuando se inicia la sesión con el programa, se deberá presionar el botón central del arreglo de botones de control del punto de vista; es decir, el botón con un punto negro  . Al presionar dicho botón, el programa, de forma semejante a como lo hace cuando se presiona un botón de vistas predeterminadas, calculará la ruta a seguir para lograr orientar el punto de vista a su origen a partir del punto de vista actual y por medio de quaterniones hará una interpolación para encontrar 200 orientaciones intermedias para lograr una transición de la escena en forma suavizada, hasta llegar a la orientación del punto de vista deseado.

7.3. Teach pendant

El programa cuenta con un emulador del teach pendant con el que normalmente viene equipado el robot CRS A465 (ver Figura 58). Este emulador cuenta con un teclado mostrado en la interfase del usuario; y cuenta con las mismas teclas que tiene el teach pendant original. Como se dijo anteriormente, el emulador del teach pendant cuenta también con una pantalla para mandar mensajes escritos al operador; en esta pantalla, el operador podrá ver los diferentes estados resultado de la navegación por los menús de operación del teach pendant.

El emulador del teach pendant ha sido diseñado de forma que se apegue lo más acertadamente posible a las funciones del teach pendant original; teniendo en cuenta que el simulador no cuenta con un programa para el robot como el que se puede obtener al tener un método de programación por medio de la computadora. Y tampoco cuenta con un medio de comunicación hacia el controlador del robot ni al robot mismo físicamente; entonces, por ejemplo, no se puede hacer movimientos físicos al robot y obtener un resultado de la posición del robot en la pantalla del emulador, que es lo que normalmente se hace al desactivar uno o todos los motores de las articulaciones para mover el robot manualmente hasta la posición deseada para ser grabada en una variable de posición.

7.3.1. Teclado

El teclado cuenta con varios grupos de teclas, clasificadas según sus aplicaciones; a saber, teclas de función, teclas de ejes, teclas de movimiento, teclas de arreglos y teclas para datos. En la Figura 59 se muestra una imagen réplica del teclado del teach pendant.

Figura 58. Teach pendant del robot CRS A465

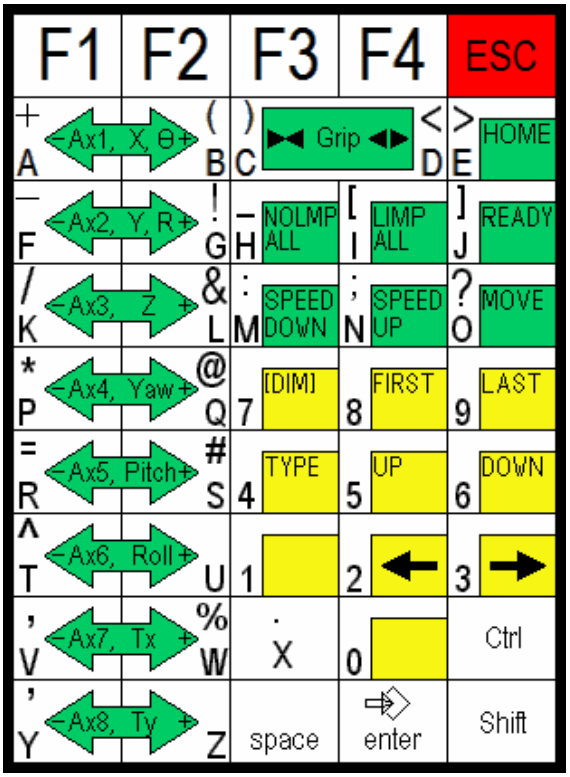
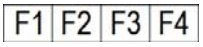


Figura 59. Imagen réplica del teclado del teach pendant

7.3.1.1. Teclas de función

Las teclas de función, 

sirven para seleccionar opciones de los diferentes menús. Tales opciones cambian dependiendo el menú actual.

Dentro de las teclas de función se encuentra la tecla de escape 

, la cual sirve para saltar a un menú de mayor jerarquía; o bien, para salir de un submenú al submenú o menú anterior, según sea el caso.

7.3.1.2. Teclas de ejes

Existen algunas teclas que sirven para mover los valores de las coordenadas de posicionamiento del robot u orientación del órgano efector. El tipo de coordenada a variar dependerá del tipo de coordenadas que se encuentren activas en el programa; como pueden ser, coordenadas de articulaciones o coordenadas cilíndricas por ejemplo. En la sección 7.5 Sistemas de coordenadas, se explican los tipos de sistemas de coordenadas que existen.

Las teclas 

sirven para incrementar o decrementar el valor del ángulo de rotación para la articulación de la cintura del robot, conocido como θ_1, en caso de encontrarse trabajando con coordenadas de articulaciones (JOINT). En caso de estar trabajando con coordenadas cilíndricas (CYL), estas mismas teclas servirán para incrementar o decrementar el valor del ángulo de rotación θ. Para el caso de las coordenadas universales (WORLD) o de herramienta (TOOL), estas teclas servirán para variar el valor de la coordenada X.

Las teclas  sirven para variar el valor del ángulo de rotación para la articulación del hombro del robot, conocido como θ_2, en caso de encontrarse trabajando con coordenadas de articulaciones (JOINT). En el caso de las coordenadas cilíndricas (CYL), servirán para cambiar el valor del radio o distancia a partir del eje Z. Para el caso de las coordenadas universales (WORLD) o de herramienta (TOOL), estas teclas servirán para variar el valor de la coordenada Y.

Las teclas  a su vez, en caso de encontrarse trabajando con coordenadas de articulaciones (JOINT), sirven para variar el valor del ángulo de rotación para la articulación del codo del robot, conocido como θ_3. En el caso de las coordenadas cilíndricas (CYL), coordenadas universales (WORLD) o de herramienta (TOOL), estas teclas servirán para cambiar el valor de la coordenada Z.

Las teclas  ,  y  sirven para variar las componentes de alabeo (yaw), cabeceo (pitch) y giro (roll); es decir, mueven las tres articulaciones del robot que definen la orientación del órgano efector o herramienta.

7.3.1.3. Teclas de movimiento

Existen dos teclas que permiten mover la posición y orientación del robot desde cualquier otra posición en el espacio de trabajo, hacia diferentes posiciones y orientaciones predefinidas (ver también la sección 7.6.2 Variables de posición).

La tecla  sirve para mandar el robot desde cualquier posición u orientación hacia la posición de inicio (READY); la cual consiste en aquella con los valores de 0º, 0º, -90º, 0º, 0º y 0º para los ángulos de rotación θ_1, θ_2, θ_3, θ_4, θ_5 y θ_6 respectivamente (ver Figura 48).

La tecla  sirve para mover el robot desde cualquier posición u orientación hacia la posición y orientación definida por una variable de posición previamente seleccionada (ver sección 7.6.2 Variables de posición).

7.3.1.4. Teclas de arreglos

Al estar operando el programa simulador del robot, es posible hacer que el robot se mueva a una posición previamente grabada; y al tener dos o más posiciones ya grabadas, es posible que el robot se mueva a cualquiera de ellas. Para hacer esto es necesario seleccionar primeramente la variable de posición que corresponda con la posición y orientación deseada, para luego ordenar al robot a tomar tal posición y orientación. Para navegar entre las posibles variables de posición previamente grabadas, se proporcionan cuatro teclas del teach pendant. Estas teclas son: para ir a la primer variable de posición del listado de variables existentes, se puede hacer clic sobre la tecla  ; para ir a la última, la tecla  ; para avanzar una variable hacia delante, la tecla  ; y para retroceder una variable hacia atrás, la tecla  .

7.3.1.5. Teclas para datos

Las teclas para datos son aquellas que sirven para introducir caracteres al estar escribiendo el nombre de una variable por ejemplo (ver sección 7.6.2 Variables de posición). Estas teclas son todas aquellas que contienen en una de las esquinas inferiores una de las letras del abecedario o un número del 0 al 9; es decir, todas las teclas del teclado excepto las teclas de función y las teclas  ,  y  . La tecla  sirve para que cuando se quiera anexar una letra en el nombre de una variable, ésta sea escrita como minúscula.

7.3.2. Menús

La operación del simulador del robot se maneja de forma semejante a como se hace con el teach pendant original para operar el robot físicamente; esto es por medio de menús de operación. El comportamiento de las teclas del teach pendant simulado depende del menú que se encuentre activo en el programa. El operador podrá navegar en los diferentes menús del programa por medio del mismo teclado. El menú actual en que se encuentre el programa será desplegado en la pantalla del teach pendant simulado. En la Figura 60 se muestra un bosquejo de los menús del programa.

7.4. Moviendo el robot

Los movimientos del robot en el simulador solamente pueden ser generados por medio de movimiento manuales utilizando el teclado del teach pendant simulado; estos movimientos pueden ser por ejemplo, al seguir a una posición previamente grabada (ver sección 7.6.2.3 Grabar una posición y orientación en una variable); o bien, haciendo variar las coordenadas en uno de los cuatro diferentes sistemas de coordenadas que existen; a saber, coordenadas de articulaciones, coordenadas cilíndricas, coordenadas universales y coordenadas de herramienta.

7.5. Sistemas de coordenadas

Dentro del sistema del robot, los cuatro sistemas de coordenadas que existen se conocen como modos; cada modo corresponde exclusivamente a cada uno de los sitemas de coordenadas y cada modo define la forma en que el robot responde a cada una de las teclas de ejes.

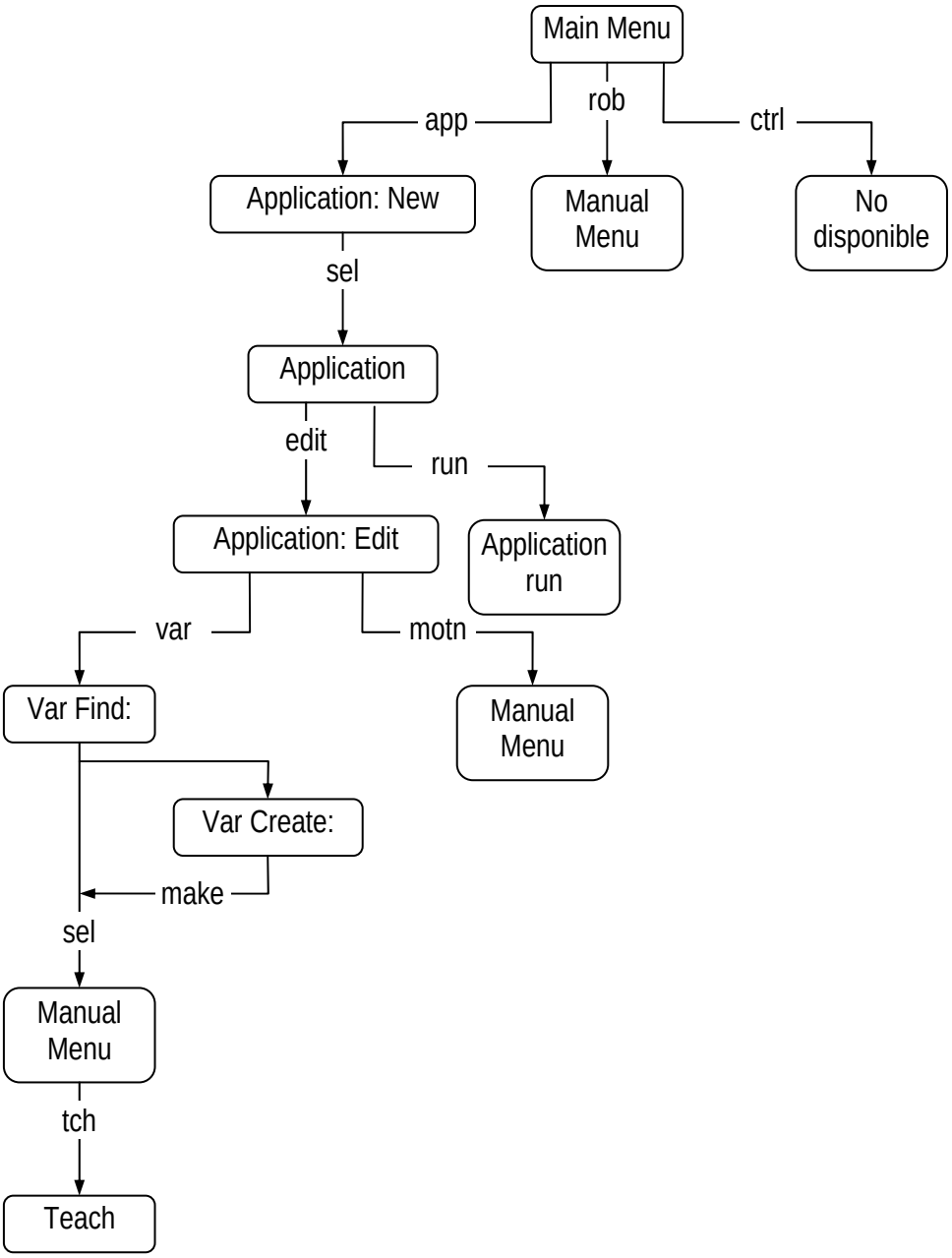


Figura 60. Diagrama jerárquico de los menús del simulador

7.5.1. Coordenadas de articulaciones (JOINT)

El primer sistema de coordenadas se conoce como coordenadas de articulaciones o modo JOINT. Este sistema de coordenadas es un sistema rotacional, dado que las coordenadas se dan en función de giros con respecto a los ejes de rotación de las articulaciones. Cada eje de rotación pasa a través de la articulación y es el centro de giro de esa articulación. En este sistema de coordenadas cada una de las teclas de ejes hace girar una de las articulaciones. En la Figura 48 se muestran las articulaciones del robot CRS A465. Para la articulación 1, la rotación positiva corresponde a una rotación con respecto al eje Z de las coordenadas universales. Para las articulaciones 2, 3 y 5, la rotación positiva es aquella que levanta y aleja el órgano efector cuando éste se encuentra hacia el frente del robot (el frente del robot se considera hacia donde apunta el eje X del sistema de coordenadas universales). Para las articulaciones 4 y 6, la rotación positiva es aquella que corresponde a una rotación con respecto al eje Z de las coordenadas universales, si el brazo del robot estuviera en una posición apuntando hacia arriba.

Las teclas  sirven para incrementar o decrementar el valor del ángulo de rotación para la articulación de la cintura del robot, conocido como θ_1.

Las teclas  sirven para variar el valor del ángulo de rotación para la articulación del hombro del robot, conocido como θ_2.

Las teclas  a su vez, sirven para variar el valor del ángulo de rotación para la articulación del codo del robot, conocido como θ_3.

Las teclas  ,  y  sirven para variar las componentes de alabeo (yaw), cabeceo (pitch) y giro (roll); es decir, mueven las tres articulaciones del robot que definen la orientación del órgano efector o herramienta.

Las coordenadas de articulaciones pueden describir cualquier posición y orientación del brazo del robot dentro de su espacio de trabajo. El punto central de la herramienta (TCP por sus siglas en inglés) se puede mover de acuerdo a los movimientos de las articulaciones; pero solamente se puede mover una articulación a la vez y por lo tanto, no se pueden realizar movimientos en línea recta.

Las coordenadas completas se escriben en grados en el orden θ_1, θ_2, θ_3, θ_4, θ_5, θ_6.

Para efecto de análisis, los valores de rotación de las articulaciones que definen la posición y orientación del robot siempre estarán disponibles en la pantalla de la interfase del usuario, dentro del recuadro que se encuentra en la parte inferior derecha.

7.5.2. Coordenadas cilíndricas (CYL)

Este tipo de coordenadas se basa en un eje vertical llamado eje Z que coincide con el eje Z del sistema de coordenadas universales. Las coordenadas cilíndricas se definen como:

a) Un ángulo de rotación alrededor del eje Z, llamado θ. Este ángulo es el mismo ángulo de rotación de la cintura del robot θ_1.

b) Una distancia a partir del eje Z hacia afuera, llamada radio R.

c) Una distancia vertical a lo largo del eje Z, llamada altura Z.

Las coordenadas cilíndricas pueden describir cualquier posición del punto central de la herramienta TCP dentro del espacio de trabajo de robot. La orientación del órgano efector es descrita por rotaciones positivas o negativas con respecto a unos ejes basados en las coordenadas cilíndricas; La rotación alrededor del eje R, sería el giro (roll), una rotación sobre un eje tangente al radio R y al ángulo de rotación θ sería el cabeceo (pitch) y una rotación sobre el eje Z sería el alabeo (yaw).

Las coordenadas completas se escriben en el orden θ, R, Z, yaw, pitch, roll. Los ángulos de rotación se escriben en grados.

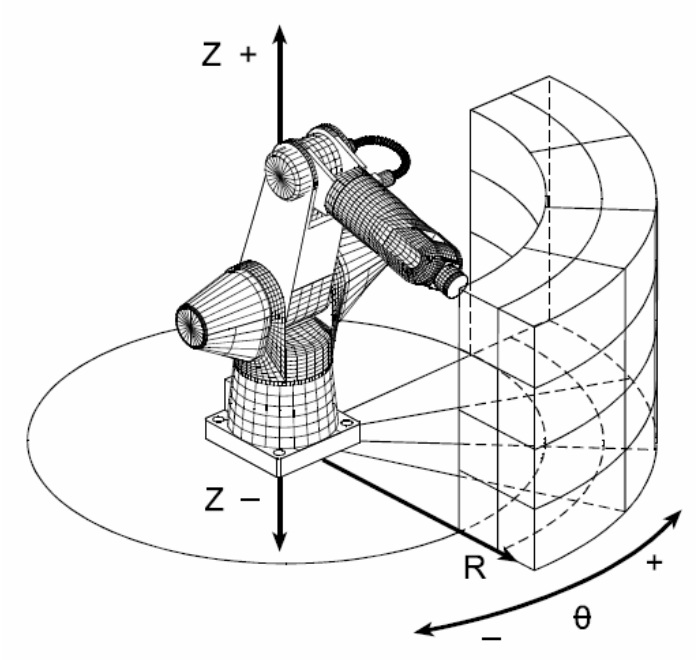


Figura 61. Coordenadas cilíndricas (CYL)

7.5.3. Coordenadas universales (WORLD)

Este tipo de coordenadas son las comúnmente conocidas como coordenadas cartesianas en tres dimensiones; se basan en tres ejes mutuamente perpendiculares identificados como eje X, eje Y y

eje Z. El eje X se encuentra orientado hacia enfrente del robot, el eje Y se encuentra orientado hacia la izquierda del robot y el eje Z hacia arriba. El origen del sistema se encuentra en el centro del plano inferior de la base del robot.

Las coordenadas universales pueden describir cualquier posición del punto central de la herramienta TCP dentro del espacio de trabajo de robot. La orientación del órgano efector es descrita por rotaciones positivas o negativas con respecto a unos ejes basados en las coordenadas universales; La rotación alrededor del eje X, sería el giro (roll), una rotación sobre el eje Y sería el cabeceo (pitch) y una rotación sobre el eje Z sería el alabeo (yaw).

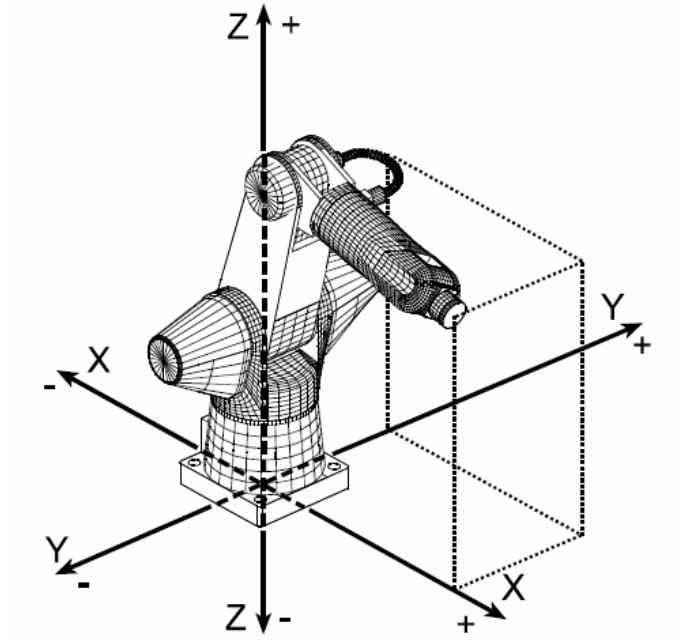


Figura 62. Coordenadas universales (WORLD)

Las coordenadas completas se escriben en el orden X, Y, Z, yaw, pitch, roll. Los ángulos de rotación se escriben en grados.

7.5.4. Coordenadas de herramienta (TOOL)

De forma semejante al sistema de coordenadas universales, el sistema de coordenadas de herramienta esta basado en tres ejes X, Y y Z mutuamente perpendiculares que forman un sistema del tipo dextrogiro. El origen del sistema se encuentra localizado en el punto central de la herramienta TCP y por lo tanto se mueve junto con la herramienta. El eje X se conoce también como eje de la herramienta y se encuentra orientado en forma perpendicular y hacia afuera del plano de la herramienta. Los ejes Y y Z se encuentran sobre el plano de la herramienta. La orientación del órgano efector es descrita por rotaciones positivas o negativas con respecto a unos ejes basados en las coordenadas universales; La rotación alrededor del eje X, sería el giro (roll), una rotación sobre el eje Y sería el cabeceo (pitch) y una rotación sobre el eje Z sería el alabeo (yaw).

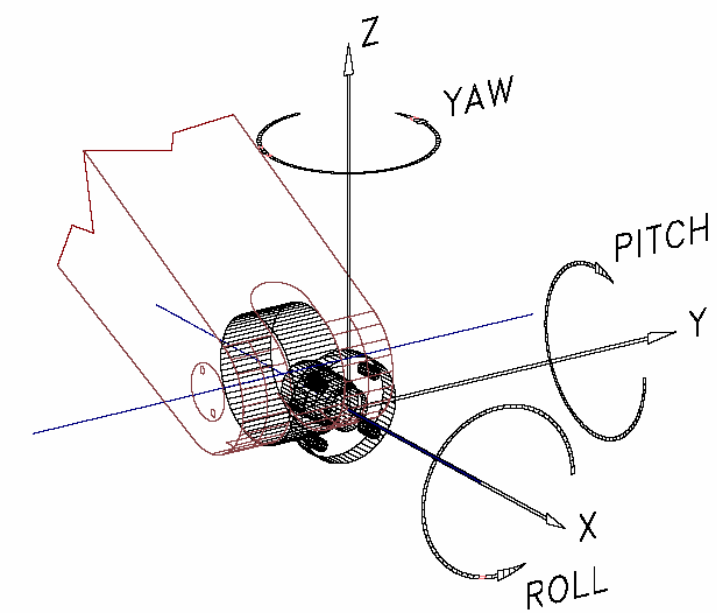


Figura 63. Coordenadas de herramienta (TOOL)

7.6. Menú de operación manual

La operación manual del robot al hacer variar las coordenadas dentro de uno de los cuatro sistemas de coordenadas descritos en la sección 7.5 Sistemas de coordenadas, se puede hacer solamente dentro del menú de operación manual. Un ejemplo de esta pantalla se presenta en la Figura 65. Existen dos caminos para llegar al menú de operación manual. El primer camino y el más sencillo es, desde el menú principal, mostrado en la Figura 64, el operador puede seleccionar la opción “rob” con la tecla F2 y el programa cambiará a la pantalla mostrada en la Figura 65. El otro camino es siguiendo el método de selección de una variable de posición, descrito en la sección 7.6.2.1 Selección de una variable de posición.

La diferencia del resultado de estos dos caminos consiste en que al seleccionar una variable de posición, el menú de operación manual ofrece al operador la capacidad de grabar una posición y orientación en la variable previamente seleccionada (ver sección 7.6.2.3 Grabar una posición y orientación en una variable).

Figura 64. Pantalla del menú principal

Figura 65. Pantalla del menú de operación manual

7.6.1. Programas del robot

El programa simulador no tiene la función de seleccionar un programa previamente creado en una computadora; sin embargo, se presenta la pantalla de selección de programa sólo para mostrar al usuario de qué forma aparecerá esta opción en la pantalla del teach pendant real como menú previo a otras pantallas con funciones importantes dentro de la manipulación del robot. Primero se muestra una pantalla para creación de un programa nuevo; tal como se muestra en la Figura 66, donde se puede modificar el nombre del programa a crear. Este nombre se perderá al salir al menú principal y regresar a la pantalla de creación de programas. Para llegar a esta pantalla desde el menú principal se debe seleccionar la opción “app” con la tecla F1.

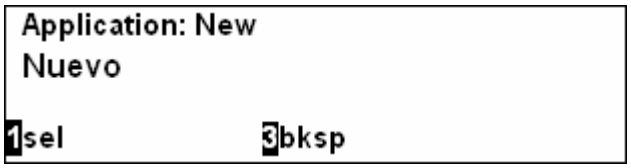


Figura 66. Pantalla de creación de un programa nuevo

Después de editar el nombre del programa deseado, se deberá seleccionar la opción “sel” con la misma tecla F1 para pasar a la pantalla del menú de programa; esta pantalla se muestra en la Figura 67, en donde se presenta la opción de correr el programa seleccionado, con la opción “run”; pero, debido a que no existe en realidad programa alguno, para efecto de demostración, en el programa de simulación se ha asignado la función de correr un programa, que no es sino un recorrido del robot a través de todas las posiciones previamente grabadas. La otra opción disponible sirve para entrar al menú de edición del programa virtual y esto se hace seleccionando la opción “edit” con la tecla F1.

Figura 67. Pantalla de menú de programas

7.6.2. Variables de posición

El programa tiene la capacidad de crear variables de posición y grabar en ellas posiciones y orientaciones del robot, para después poder mover el robot desde cualquier otra posición u orientación del brazo hacia tal posición y orientación previamente grabada.

7.6.2.1. Selección de una variable de posición

Para entrar al menú de selección de una variable de posición, se debe seguir la ruta establecida en la sección 7.6.1 Programas del robot, hasta llegar a la pantalla de menú de programas mostrada en la Figura 67, y después seleccionar la opción “edit” con la tecla F1. Después aparecerá la pantalla de selección de una variable de posición mostrada en la Figura 68. En esta pantalla se podrá navegar por la lista de variables previamente creadas para ser seleccionada o borrada según sea el caso. En la pantalla del ejemplo se muestra una variable del tipo cloc (Cartesian LOCation) llamada “ABC”, la cual se encuentra en blanco; es decir no se ha grabado ninguna posición en ella. Esto se sabe al encontrar el mensaje “(undef)” en la parte superior derecha de la pantalla.

En caso de existir más de una variable de posición creada en la lista de variables disponibles, el operador podrá navegar a través de esta lista seleccionando las opciones “prev” o “next”, con las teclas F3 o F4 respectivamente.

Si se quiere borrar una de las variables de la lista de variables disponibles, el operador después de haberla encontrado al navegar por el listado, podrá borrarla seleccionando la opción “del” con la tecla F2.

Después de encontrar la variable deseada para seleccionar, ya sea para grabar una posición en ella; o bien, para mover el robot hacia la posición previamente grabada correspondiente con la variable, el operador podrá seleccionar la variable al escoger la opción “sel” con la tecla F1. Después el programa cambiará a la pantalla del menú de movimiento manual del robot semejante a la mostrada en la Figura 71.

Figura 68. Pantalla de selección de una variable de posición

7.6.2.2. Crear una variable de posición

Estando en la pantalla de selección de una variable, como la mostrada en la Figura 68, para crear una variable y anexarla al listado de variables disponibles, el operador puede comenzar a escribir el nombre deseado para la variable nueva utilizando las teclas de datos (ver sección 7.3.1.5 Teclas para datos); en este momento la pantalla del teach pendant simulado cambia a la pantalla de creación de variables, mostrada en la Figura 69. El operador podrá escribir el nombre deseado para la variable a crear utilizando las teclas de datos. En caso de querer utilizar una letra minúscula, se deberá seleccionar la tecla Shift antes de seleccionar la tecla con la letra que se desea minúscula. Si

se quiere borrar el último carácter escrito en el nombre de la variable, el operador podrá seleccionar la opción “bksp” con la tecla F4.

El operador deberá seleccionar el tipo de variable a crear. Al seleccionar la opción “type” con la tecla F2, el operador podrá navegar entre los tipos de datos disponibles. Para efectos de simulación, el tipo de datos seleccionado no afecta la forma en que se grabara la información en la variable. Por lo anterior, es irrelevante el tipo de variable seleccionado para la variable a crear; sin embargo, se presenta en la simulación para efectos educativos solamente. A pesar de que el tipo de variable es irrelevante, aún así el operador deberá seleccionar un tipo de variable para poder crearla.

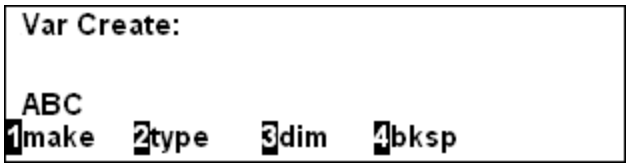


Figura 69. Pantalla 1 de creación de una variable

De igual forma, en el simulador se presenta la opción “dim” sin efecto alguno sobre la variable a crear; opción que normalmente sirve para crear variables dentro de un arreglo de dos o más dimensiones.

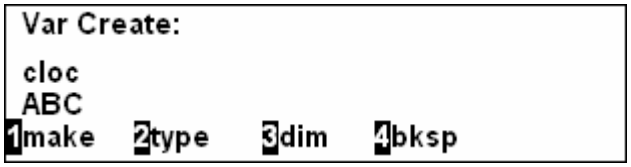


Figura 70. Pantalla 2 de creación de una variable

Una vez satisfecho con el nombre y tipo de variable a crear, el operador deberá seleccionar la opción "make" con la tecla F1. En este momento la variable se anexa al final del listado de variables disponibles y se encuentra lista para ser seleccionada y poder grabar una posición y orientación del robot en ella.

7.6.2.3. Grabar una posición y orientación en una variable

Estando en la pantalla de selección de una variable mostrada en la Figura 68, una vez que el operador encuentre la variable deseada sobre la que se quiere grabar una posición y orientación, deberá seleccionar la variable escogiendo la opción "sel" con la tecla F1. Después el programa cambiará a la pantalla del menú de movimiento manual del robot semejante a la mostrada en la Figura 71. En este momento, el operador puede mover el robot hacia cualquier posición y orientación que desee grabar; esto es siguiendo el método de movimiento manual del robot descrito en la sección 7.6 Menú de operación manual. Una vez que el operador esté satisfecho con la posición y orientación del robot que quiera grabar, para almacenar los datos de la posición y orientación necesarios en la variable previamente seleccionada, el operador deberá seleccionar la opción "tch" con la tecla F1.

Figura 71. Pantalla de movimiento manual del robot con opción de grabar

7.6.2.4. Moviendo el robot a través de un arreglo de posiciones

Estando en el menú de operación manual, el operador podrá mover el robot del simulador hacia una variable previamente grabada (ver sección anterior), tal como se explicó en la sección 7.6 Menú de operación manual. Además, ya estando en el menú de operación manual, y teniendo una lista de variables de posición previamente grabadas, el operador puede hacer que el robot siga de una posición a otra correspondiente a una variable pregrabada utilizando las teclas de arreglos (ver sección 7.3.1.4 Teclas de arreglos).

Conclusiones y recomendaciones

Se ha logrado desarrollar un programa de simulación para representar la cinemática del robot CRS A465 por medio del lenguaje de programación Visual Basic V 6.0, utilizando gráficos en sólidos sombreados en 3D haciendo uso de una librería de OpenGL®, modelando la cinemática del robot por medio de quaterniones.

Este programa de simulación puede ser utilizado como herramienta de simulación para facilitar la enseñanza de la robótica a los alumnos estudiantes de alguna materia relacionada con la robótica, con la ventaja de poder correr el programa sobre cualesquiera computadoras compatibles disponibles en los laboratorios de cómputo dentro de las instalaciones del ITESCA o cualquier otra institución de enseñanza superior.

La utilización de la librería OpenGL para la graficación de las imágenes facilita en gran medida la modelación de los cuerpos a proyectar en la pantalla; ésto es comparando este método con aquel utilizando la teoría explicada en el capítulo 3.

La utilización de quaterniones para modelar la cinemática del robot representa una alternativa que facilita la aplicación de rotaciones a cuerpos rígidos en el espacio, evitando también los problemas de Gimbal lock (ver sección 5.2.2 Ángulos de Euler). Además la interpolación lineal esférica con quaterniones permite al programador realizar la animación del movimiento de los objetos en la pantalla dando la apariencia de movimiento suave.

Este trabajo conforma una mínima parte de un proyecto mucho más ambicioso que cubriría las

actividades desde la generación de un programa, la interpretación de dicho programa para ser transferido al simulador, incluyendo un entorno de trabajo completo para la realización de una tarea, y por último la transferencia de dicho programa hacia un controlador que lleve las órdenes de movimiento a cada uno de los actuadores del robot físico. Para todo ello, entonces, queda pendiente para trabajo futuro el desarrollo de un intérprete o compilador de texto para el desarrollo de programas de aplicación para el robot y la interfase de comunicación para transferir los comandos de movimiento al controlador del robot para cada una de las articulaciones y actuadores involucrados en la celda de trabajo. Faltaría entonces, a su vez modificar o adaptar el simulador para permitir al usuario modelar el entorno de trabajo; se sugiere el desarrollo de una interfase para crear archivos donde se pueda almacenar la información necesaria para manipular diferentes ambientes de trabajo con diversos componentes, de forma similar a aquel que ofrecen los simuladores RoboWorks (ver sección 1.2 Simulador RoboWorks) y Ropsim (ver sección 1.3 Simulador Ropsim).

Otro trabajo a futuro recomendado sería aquel relacionado a la modificación del simulador para presentar resultados de la cinemática del robot incluyendo análisis de velocidad y aceleración; así como la dinámica del mismo.

Bibliografía

[Animation] Animation of rigid bodies

http://citeseer.ist.psu.edu/cache/papers/cs/21004/http:zSzzSzwww.cg.tuwien.ac.atzSzcourseszSzAnimationzSzD_Rigid_Bodies.pdf/animation-of-rigid-bodies.pdf

[Apollo 11] Apollo 11. Lunar Surface Journal

http://www.hq.nasa.gov/office/pao/History/alsj/a11/a11.postland.html

[Barr, 1992] Barr, A.H., et al. "Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions". Computer Graphics 26,2 (July 1992), 313-320.

http://citeseer.ist.psu.edu/cache/papers/cs/651/ftp:zSzzSzftp.gg.caltech.eduzSzpubzSzPaperszSzbarr_siggraph92.pdf/barr92smooth.pdf

[Bearlocher, 2000] Baerlocher, P., Boulic, R.: Parametrization and range of motion of the ball-and-socket joint. Proceedings of AVATARS'2000 Conference, Lausanne (2000) 180–190

http://citeseer.ist.psu.edu/cache/papers/cs/25914/http:zSzzSzligwww.epfl.chzSz~bouliczSzPublications_htmlzSz2000zSzAVATAR.pdf/baerlocher00parametrization.pdf

[Bearlocher, 2001] Bearlocher, P. Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures. Tesis doctoral, EPFL (2001)

http://ligwww.epfl.ch/~baerloch/papers/thesis.pdf

[Bru, 2001] Bru Rafael, Climent Joan-Josep, Mas Josep, Urbano Ana. Álgebra Lineal. Universidad Politécnica de Valencia. Alfaomega Grupo Editor S. A. de C. V. ISBN 970-15-0660-X. México, D. F. 2001

[CECIC, 2002] Programa Regional de Competitividad Sistémica. Programa elaborado por el Centro de Capital Intelectual y Competitividad (CECIC) para el Gobierno del Estado de Coahuila Mayo,2002:

http://www.contactopyme.gob.mx/regional/

[Comrade] Comrade J/SAE, Carl-Henrik Skårstedt. How to code vectordemos. http://www.mways.co.uk/amiga/howtocode/text/vectors.php

[CPEN, 1997] CPEN 461/661, 1997. OpenGL Tutorial

http://www.eecs.tulane.edu/www/Terry/OpenGL/Introduction.html#Introduction

[Crews] Travis Crews. Rotational Matrices in Aeronautical Motion, the Problem of the Gimbal Lock.

http://www.mathsci.appstate.edu/~sjg/class/2240/finalss04/Travis.html

[Dam, 1998] E. B. Dam, M. Koch, and M. Lillholm, Quaternions, interpolation and animation, Tech. Rep. DIKU 98/5, Institute of Computer Science, Univ. Of Copenhagen, 1998.

http://www.diku.dk/students/myth/quat.html

[DeRose, 1992] DeRose, Tony D. Three-Dimensional Computer Graphics, A Coordinate-Free Approach. University of Washington. 1992. http://mrl.snu.ac.kr/CourseDataDrivenAnimation/readings/derose92threedimensional.pdf

[Encarnação] Página web del simulador Encarnação

http://www.encarnacao.com/e_index.htm

[Encyclopedia] http://encyclopedia.laborlawtalk.com/Euler_angles

[Epstein, 1989] D. Epstein, F. Jansen, and J. Rossignac. Zbuffer rendering from CSG: The trickle algorithm. IBM Research Report RC 15182, Nov 1989.

[Erhart, 2000] ERHART, G., AND TOBLER, R. General purpose z-buffer CSG rendering with consumer level hardware. Tech. Rep. VRVis 003, VRVis Zentrum f¨ur Virtual Reality und Visualisierung Forschungs-GmbH, 2000. http://www.vrvis.at/TR/2000/TR_VRVis_2000_003_Full.pdf

http://citeseer.ist.psu.edu/cache/papers/cs/25682/http:zSzzSzwww.vrvis.atzSzresearchzSz..zSzTRzSz2000zSzTR_VRVis_2000_003_Full.pdf/erhart00general.pdf

[Esteve, 2001] D. Juan Domingo Esteve. Apuntes de Robótica. Instituto de Robótica de la Universistat de Valencia (España). 3 de Febrero de 2001

http://www.ingenieroseninformatica.org/recursos/tutoriales/aprob/index.php

[Fang, 2000] Shiaofen Fang and Duoduo Liao, "Fast CSG Voxelization by Frame Buffer Pixel Mapping", ACM/IEEE Volume Visualization and Graphics Symposium 2000 (Volviz'00), pp 43-48, Salt Lake City, UT, 9-10 October 2000. Shiaofen Fang, Duoduo Liao.

http://www.cs.iupui.edu/~sfang/volvis00.pdf

[FEM] Página web del Foro Económico Mundial:

http://www.weforum.org/

[Goldfeather, 1986] Goldfeather, Jack, Jeff P. M. Hulquist y Henry Fuchs, “Fast Constructive Solid Geometry Display, in the Pixel-Powers Graphics System”, en Proceedings of the 13th annual conference on computer Graphics and interactive techniques. ACM Press, 1986, pp. 107-116. http://www.cs.unc.edu/~fuchs/publications/FastConstructSolid86.pdf

[Goldfeather, 1989] Goldfeather, J., S. Molnar, G. Turk, y H. Fuchs. "Near Real-time CSG Rendering using Tree Normalization and Geometric Pruning," IEEE Computer Graphics and Applications , 9(3), Mayo 1989, 20-28. http://www.cc.gatech.edu/~turk/my_papers/pxpl_csg.pdf

[Grassia, 1998] Grassia, Sebastian F. Practical Parameterization of Rotations Using the Exponential Map. Carnegie Mellon University. The Journal of Graphics Tools, volume 3.3, 1998.

http://mrl.snu.ac.kr/CourseDataDrivenAnimation/readings/expmap.pdf

[Gruber, 2000] Gruber, Diana. Do we really need quaternions?. 2000

http://www.gamedev.net/reference/articles/article1199.asp

http://www.gamedev.net/community/forums/topic.asp?topic_id=25314

[Guha] Sudipto Guha, Shankar Krishnan, Kamesh Munagala and Suresh Venkatasubramanian. CSG Rendering and the Computational Power of Two-Sided Depth Tests on the GPU

http://www.research.att.com/areas/visualization/gpgpu/picking/csg-full.pdf

[Guha, 2002] Sudipto Guha, Shankar Krishnan, Kamesh Munagala and Suresh Venkatasubramanian. 2002. The power of a two-sided depth test and its application to csg rendering and depth extraction. Tech. Rep. TD-5FDS6Q, AT&T.

http://www.research.att.com/~krishnas/MY_PAPERS/i3d03_csg.pdf

[Guha, 2003] Sudipto Guha, Shankar Krishnan, Kamesh Munagala and Suresh Venkatasubramanian. Application of the Two-Sided Depth Test to CSG Rendering. ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics, 177-180, 2003.

http://www.research.att.com/~suresh/papers/csg/csg.pdf

[Guthrie, 1886] Guthrie Tait, Peter. Encyclopedia Britannica, Ninth Edition, 1886, Vol. XX, pp. 160-164.

[Hamilton, 1844] William Rowan Hamilton. ON A NEW SPECIES OF IMAGINARY QUANTITIES CONNECTED WITH A THEORY OF QUATERNIONS. Proceedings of the Royal Irish Academy, 2 (1844), 424-434.

http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/Quatern1/Quatern1.pdf

[Hamilton, 1844-1850] William Rowan Hamilton. ON QUATERNIONS, OR ON A NEW SYSTEM OF IMAGINARIES IN ÁLGEBRA. Philosophical Magazine, (1844–1850).

http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/OnQuat/OnQuat.pdf

[Hamilton, 1847] William Rowan Hamilton. ON QUATERNIONS. Proceedings of the Royal Irish Academy, 3 (1847), pp. 1-16.

http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/Quatern2/Quatern2.pdf

[Hamilton, 1848] William Rowan Hamilton. RESEARCHES RESPECTING QUATERNIONS: FIRST SERIES. Transactions of the Royal Irish Academy, vol. 21, part 1 (1848), pp. 199–296.

http://www.maths.tcd.ie/pub/HistMath/People/Hamilton/ResQuat/ResQuat.pdf

[Hart, 1994] J.C. Hart and G.K. Francis, Visualizing Quaternion Rotation. ACM Transactions on Graphics.; 13(3): (1994) 256-276.

http://citeseer.ist.psu.edu/cache/papers/cs/175/http:zSzzSzwww.eecs.wsu.eduzSz~hartzSzpaperszSzvqr.pdf/hart93visualizing.pdf

[Hoag, 1963] David Hoag. Apollo Guidance and Navigation Considerations of Apollo IMU Gimbal Lock. MIT Instrumentation Laboratory Document E-1344. April 1963

http://www.hq.nasa.gov/office/pao/History/alsj/e-1344.htm

[Horn, 1987] Horn, B.K.P. "Closed-form solution of absolute orientation using unit quaternions". J. Opt. Soc. Am. A 4,4 (April 1987), 629-642.

http://people.csail.mit.edu/u/b/bkph/public_html/papers/Absolute-OPT.pdf

[ITESCA] Centro de Tecnología Avanzada (CETA)del Instituto Tecnológico Superior de Cajeme (ITESCA).

(http://www.itesca.edu.mx/investigacion/investigacion.asp#)

[Jaspe, 2005] Alberto Jaspe Villanueva, Julián Dorado de la Calle. Una aproximación a OpenGL.

http://rnasa.tic.udc.es/gc/Tutorial%20OpenGL/index.htm

[Johnson, 2003] Johnson, Michael Patrick. Exploiting Quaternions to Support Expressive Interactive Character Motion. Submitted to the Program of Media Arts and Sciences in partial fulfillment of the requirements for the Degree of Doctor in Philosophy at the Massachusetts Institute of Technology. February 2003.

http://characters.media.mit.edu/Theses/johnson_phd.pdf

[Kilgard] Kilgard, Mark, "Fast OpenGL-rendering of Lens Flares", http://reality.sgi.com/mjk/tips/lensflare/

[Kilgard, 1996] Mark J. Kilgard. The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3. Silicon Graphics, Inc. 13 Noviembre de 1996.

http://www.opengl.org/documentation/specs/glut/glut-3.spec.pdf

[Kirsch, 2004] F. Kirsch, J. Döllner, Rendering Techniques for Hardware-Accelerated Image-Based CSG Journal of WSCG 2004, Vol. 12, No. 2, 2004, pp. 221-228.

http://wscg.zcu.cz/wscg2004/Papers_2004_Full/M11.pdf

[Márquez, 2000] Márquez Miranda Mario. Modelado cinemático y dinámico de robots utilizando Quaterniones. Tesis de Doctorado en Ingeniería, Universidad Anáhuac del Sur, México, D. F. 2000.

[McReynolds, 1998] McReynolds, Tom, David Blythe, Brad Grantham, and Scott Nelson, "Programming with OpenGL: Advanced Techniques", Course 17 notes at SIGGRAPH 98, 1998. http://reality.sgi.com/blythe/sig99/index.html

[McReynolds, 1998-2] Tom McReynolds y David Blythe, Advanced Graphics Programming Techniques Using OpenGL. 1998.

http://www.opengl.org/resources/tutorials/advanced/advanced98/notes/notes.html

[Moreno, 2004] M. A. Moreno Armendáriz, J. D. Peña Benítez y C. P. Salinas Mendoza. Teach Pendant Simulator for Education in Robotics. International Simposium on Robotics and Automation 2004. August 25-27, 2004. Querétaro. México

http://www.ci.ulsa.mx/~mmoreno/pdf/articulos/ISRA2004.pdf

[Mukundan, 2002] R. Mukundan. Quaternions: From Classical Mechanics to Computer Graphics, and Beyond. Department of Computer Science University of Canterbury Christchurch, New Zealand. Proceedings of the 7th Asian Technology Conference in Mathematics 2002.

http://www.cosc.canterbury.ac.nz/people/mukundan/atcm02_1.pdf

[Munagala, 2003] K. Munagala, S. Guha, S. Krishnan, S. Venkat, The Power of a Two-Sided Depth Test and its Application to CSG Rendering, Proceedings of ACM Siggraph Symposium on Interactive 3D Graphics, 2003

http://citeseer.ist.psu.edu/cache/papers/cs/27402/http:zSzzSzwww.stanford.eduzSz~kameshzSzcsg.pdf/the-power-of-a.pdf

[Myung-Soo] Myung-Soo Kim y Kee-Won Nam. A C2-contonuous B-Spline Quaternion Curve Interpolating a Given Sequence of Solid Orientations.

http://citeseer.ist.psu.edu/cache/papers/cs/11733/http:zSzzSzwww.cgvr.postech.ac.krzSzmskimzSzftpzSzca95.pdf/kim95ccontinous.pdf

[Myung-Soo, 1995] Myung-Soo Kim y Kee-Won Nam. Interpolating Solid Oreintations with Circular Blending Quaternion Curves. Computer Aided Design, Vol 27. No. 5. pp. 385-398, 1995.

http://citeseer.ist.psu.edu/cache/papers/cs/11733/http:zSzzSzwww.cgvr.postech.ac.krzSzmskimzSzftpzSzcad95b.pdf/kim95interpolating.pdf

[Myung-Soo, 1995-2] Myung-Soo Kim y Kee-Won Nam. Hermite Interpolation of Solid Orientations Based on a Smooth Blending of two Great Circular Arcs on SO(3). In computer Graphics, Development in Virtual Enviroments. R. Earnshaw and J. Vince (eds.) Academic Press, pp. 171-183, 1995.

http://citeseer.ist.psu.edu/cache/papers/cs/11733/http:zSzzSzwww.cgvr.postech.ac.krzSzmskimzSzftpzSzcgi95.pdf/nam95hermite.pdf

[National Instruments] Página web de National Instruments

http://www.ni.com/labview/

[OpenGL] OpenGL Overview

http://www.opengl.org/about/overview.html

[OpenGL-2] OpenGL Architecture Review Board. OpenGL Reference Manual (second edition). Addison-Wesley, 1996. ISBN 0-201-46140-4.

http://www.opengl.org/documentation/blue_book_1.0/

[O'Sullivan, 1995] Bryan O'Sullivan. Lazy functional quaternions. May 28, 1995

http://citeseer.ist.psu.edu/cache/papers/cs/17171/ftp:zSzzSzftp.maths.tcd.iezSzpubzSzbosullvnzSzpaperszSzhaskell-quaternions.pdf/lazy-functional-quaternions.pdf

[Palomares, 2006] Palomares, Juan Enrique. Modelación cinemática del Robot CRS A465, utilizando el álgebra de quaterniones. Tesis de Maestría en Ingeniería Mecánica. Universidad Nacional Autónoma de México. Facultad de Ingeniería. 2006

[Pawasauskas, 1997] John Pawasauskas. Volume Visualization With Ray Casting. CS563 - Advanced Topics in Computer Graphics February 18, 1997

http://www.cs.wpi.edu/~matt/courses/cs563/talks/powwie/p1/ray-cast.htm

[Permadi] F. Permadi. Ray-Casting Tutorial.

http://www.permadi.com/tutorial/raycast/

[Pervin] Edward Pervin y Jon A. Webb. Quaternions in Computer Vision and Robotics.

http://citeseer.ist.psu.edu/cache/papers/cs/3716/http:zSzzSzwww.cs.berkeley.eduzSz~laurazSzcs184zSzquatzSzquats-vision-robotics.pdf/quaternions-in-computer-vision.pdf

[Peterson, 2003] I.R. Peterson, The Gibbs Representation of 3D Rotations. Centre for Molecular and Biomolecular Electronics, Coventry University, NES, Priory Street, Coventry, CV1 5FB, UK. 2003.

http://arxiv.org/ftp/cs/papers/0104/0104016.pdf

[Podobedov] Roman Podobedov. What is OpenGL.

http://www.alexmw.km.ru/opengldoc1.html

[Popov] Peter Popov, API OpenGL 2.0 Review.

http://www.digit-life.com/articles/opengl20/

[Rappoport, 1997] A. Rappoport and S. N. Spitz, "Interactive Boolean Operations for Conceptual Design of 3-D Solids", In Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, pp. 269-278, August 1997.

http://www.cs.huji.ac.il/~arir/cda.pdf

[RoboWorks] Página web del simulador RoboWorks

http://www.newtonium.com/

[Rodríguez, 2004] Luis Rodríguez Valencia. Mecánica Clásica. Departamento de Física. Universidad de Santiago de Chile. 15 de Marzo de 2004.

http://fisica.usach.cl/~lhrodrig/mecanica.pdf

[Ropsim] Página web del simulador Ropsim:

http://www.camelot.dk/english/downloadropsim.htm

[Rossignac, 1999] Jarek R. Rossignac and Aristides A. G. Requicha. Solid modeling. In J. Webster, editor, Encyclopedia of Electrical and Electronics Engineering. Webster, John Wiley & Sons,1999.

http://citeseer.ist.psu.edu/cache/papers

http://www.gvu.gatech.edu/~jarek/papers/SolidModelingWebster.pdf

[Salamin, 1979] Salamin, Eugene. "Application of Quaternions to Computation with Rotations". Internal Working Paper, Stanford AI Lab., 1979.

http://people.csail.mit.edu/u/b/bkph/public_html/courses/Articles/stanfordaiwp79-salamin.pdf

[Segal, 1999] Mark Segal, Kurt Akeley. The OpenGL® Graphics System: A Specification (Version 1.2.1). Silicon Graphics, Inc. April 1, 1999

http://www.opengl.org/documentation/specs/version1.2/opengl1.2.1.pdf

[Shoemake, 1985] Shoemake, Ken, "Animating Rotation with Quaternion Curves," Computer Graphics (SIGGRAPH '85 Proceedings), vol. 19, no. 3, pp. 245-254, July 1985.

http://www-2.cs.cmu.edu/~kiranb/animation/p245-shoemake.pdf

[Shoemake, 2002] Ken Shoemake. Quaternions. Department of Computer and Information Science University of Pennsylvania Philadelphia

http://www.cs.wisc.edu/graphics/Courses/cs-838-2002/Papers/quatut.pdf

[Silva, 2002] Cibelle Celestino Silva and Roberto de Andrade Martins. Polar and axial vectors versus quaternions. Am. J. Phys., Vol. 70, No. 9, September 2002

http://ghtc.ifi.unicamp.br/pdf/ram-91.pdf

[Stewart, 1998] N. Stewart, G. Leach, and S. John. An improved z-buffer CSG rendering algorithm. 1998 Eurographics/SiggraphWorkshop on Graphics Hardware, pages 25–30, Aug 1998. http://www.nigels.com/research/egsggh98.pdf

[Stewart, 2000] N. Stewart, G. Leach, and S. John. A z-buffer CSG rendering algorithm for convex objects. The 8-th International Conference in Central Europe on Computer Graphics, Visualisation and Interactive Digital Media ’2000 - WSCG 2000, II:369–372, Feb 2000.

http://www.nigels.com/research/wscg2000.pdf

[Stewart, 2002] N. Stewart, G. Leach, S. John, Linear-time CSG Rendering of Intersected Convex Objects, The 10-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision '2002 - WSCG 2002, Volume II, pp. 437-444.

http://www.nigels.com/research/wscg2002.pdf

[Stewart, 2003] Stewart, N., Leach, G., and John, S. Improved CSG Rendering using Overlap Graph Subtraction Sequences. Proceedings of GRAPHITE 2003, 47-53, 2003.

http://www.nigels.com/research/graphite03/graphite03.pdf

[Taylor, 1979] Taylor, R.H. "Planning and Execution of Straight Line Manipulator Trajectories". IBM J. R&D 23,4 (July 1979), 424-436.

http://www.research.ibm.com/journal/rd/234/ibmrd2304K.pdf

[Theoharis, 2001] T. Theoharis, G. Papaioannou, and E. Karabassi. The magic of the z-buffer: A survey. The 9th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision - WSCG 2001, Feb 2001.

http://citeseer.ist.psu.edu/cache/papers/cs/22157/http:zSzzSzwscg.zcu.czzSzwscg2001 zSzPapers_2001zSzR12.pdf/theoharis01magic.pdf

[Universidad de Vigo] Universidad de Vigo. Cinemática del Robot. Curso Automatización y Robótica. Escuela Técnica Superior de Ingenieros de Minas. 2003-2004.

http://www.aisa.uvigo.es/DOCENCIA/AyRobotica/cinematica.pdf

[VBOpenGL] VBOpenGL type library v1.2

http://home.pacific.net.hk/%7Eedx/tlb.htm

[Vicci, 2001] Leandra Vicci. Averages of Rotations and Orientations in 3-space. Microelectronic Systems Laboratory. Department of Computer Science. University of North Carolina at Chapel Hill. 10 August 2001

http://citeseer.ist.psu.edu/cache/papers/cs/23077/ftp:zSzzSzftp.cs.unc.eduzSzpubzSzpublicationszSztechreportszSz01-029.pdf/vicci01averages.pdf

[Vicci, 2001-2] Leandra Vicci. Quaternions and Rotations in 3-Space: The Algebra and its Geometric Interpretation. Microelectronic Systems Laboratory, Department of Computer Science, University of North Carolina at Chapel Hill. 27 April 2001

http://citeseer.ist.psu.edu/cache/papers/cs/23077/ftp:zSzzSzftp.cs.unc.eduzSzpubzSzpublicationszSztechreportszSz01-014.pdf/quaternions-and-rotations-in.pdf

[Void, 2003] Sobeit Void. Quaternion Powers. Version 1.2, February, 2003

http://www.gamedev.net/reference/articles/article1095.asp

[Welman, 1993] Chris Welman. Inverse kinematics and geometric constraints for articulated figure manipulation. Master's thesis, Simon Frasier University, 1993.

http://graphics.ucsd.edu/courses/cse169_w04/welman.pdf

[Wiegand, 1996] T.F. Wiegand, Interactive Rendering of CSG Models, Computer Graphics Forum, Vol. 15, No. 4, Oct 1996, pp. 249-261.

http://citeseer.ist.psu.edu/cache/papers/cs/15656/ftp:zSzzSzftp.arct.cam.ac.ukzSzpubzSzcsg_paper.pdf/wiegand96interactive.pdf

[Wisnesky, 2004] Ryan J. Wisnesky. The Forgotten Quaternions. June 2004

http://www.stanford.edu/~wisnesky/TeX.pdf

[Wynn, 2001] Wynn, C. Using P-Buffers for Off-Screen Rendering in OpenGL. Technical report, Nvidia Corporation, 2001.

http://developer.nvidia.com/object/PBuffers_for_OffScreen.html

[Zachariâs, 2000] Svatopluk Zachariâs, Daniela Velichová. Projection from 4D to 3D. Journal for Geometry and Graphics Volume 4 (2000), No. 1, 55-69.

http://www.heldermann-verlag.de/jgg/jgg01_05/jgg0404.pdf

[Zeller, 2004] Zeller Cyril, Randy Fernando, Mark Harris y Matthias Wloka. Programming Graphics Hardware. NVIDIA Corporation. EUROGRAPHICS 2004

http://download.nvidia.com/developer/presentations/2004/Eurographics/EG_04_TutorialNotes.pdf

Anexos y apéndices

Apéndice A. Características físicas del Robot CRS A465

Características de desempeño

Tabla 4. Características de desempeño del Robot CRS A465

Característica	Capacidad
Capacidad de carga nominal	2 Kg
Alcance (sin herramienta)	711 mm
Alcance (con herramienta estandar)	864 mm
Precisión	± 0.05 mm
Peso	31 Kg

Área de trabajo

Tabla 5. Área de trabajo del Robot CRS A465

Articulación	Rango	Velocidad máxima
J1 (Cintura)	± 175º	180º/seg
J2 (Hombro)	± 90º	180º/seg
J3 (Codo)	± 110º	180º/seg
J4 (Alabeo de muñeca Jaw)	± 180º	171º/seg
J5 (Cabeceo de muñeca Pitch)	± 105º	173º/seg
J6 (Rotación de muñeca Roll)	± 180º	171º/seg