

**UNIVERSIDAD AUTÓNOMA METROPOLITANA
UNIDAD AZCAPOTZALCO
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA
INGENIERÍA EN COMPUTACIÓN**

PROYECTO TERMINAL

**“AMBIENTE DE PROGRAMACIÓN VISUAL PARA GRAFICAR GEOMETRÍAS
SIMPLES MEDIANTE TRASFORMACIONES ORTOGONALES CON ÁLGEBRA
GEOMÉTRICA Y ÁLGEBRA MATRICIAL”.**

ALUMNOS:

CUÉLLAR MEDINA CARLOS 204201375

GARCÍA GREGORIO JERÓNIMO 204201715

ASESOR:

**M. EN C. ARAGÓN GONZÁLEZ GERARDO
NO. ECONÓMICO 1363**

TRIMESTRE: 09-O

GRUPO: I81

MEXICO, DF., A 22 DE AGOSTO DEL 2009

Agradecimientos

Gracias mis hermanos por estar cerca y apoyarme durante mi estancia en la universidad, muchas gracias a mis padres, por su cariño y por ser la razón de mis éxitos.

A mis maestros y maestras, no solo agradezco el conocimiento que con paciencia compartieron, también agradezco la pasión que lograron inculcar en mi para realizar de corazón las tareas que ahora me hacen un Ingeniero.

Agradezco a todos los amigos y amigas que el amable destino coloco en mi camino, en cada uno de ellos encontré el apoyo y la alegría que hacen de todo recorrido una experiencia inolvidable.

Gracias al Programa de Desarrollo Profesional en Automatización y un especial agradecimiento a los maestros que lo integran, su dirección y ejemplo fueron la mayor aportación no solo en mi desarrollo profesional, sino también en mi desarrollo como persona integral.

Por último agradezco a la Universidad Autónoma Metropolitana Azcapotzalco que durante todos estos años me abrió las puertas, para ser, durante un maravilloso momento de mi vida mi “Casa Abierta al Tiempo”.

Carlos Cuéllar Medina

Agradezco al Programa de Desarrollo Profesional en Automatización que brindo el espacio y el equipo necesario para llevar a cabo este trabajo, a los profesores integrantes del Programa por su valiosa cooperación y asesoría.

Gracias Beatriz Adriana Peña Ramos por su valiosa ayuda, su entusiasmo del que me impregno y del gran amor que siempre me ha tenido, por ser mi gran pareja desde cuando éramos adolescentes, soy muy feliz contigo.

A mis padres quienes me enseñaron a ser independiente, que me brindaron su apoyo, cariño y dejaron que siempre decidiera hacer de mi vida lo que me ha gustado y con lo cual he aprendido.

Mis hermanos que fueron una guía para realizar lo que hoy he logrado, que me han ayudado y guiado en varias ocasiones de mi vida.

Y gracias a la vida por las grandes cosas que siempre ha tenido para mi, por haberme permitido realizar mis estudios en la Universidad Autónoma Metropolitana y conocer amigos con los que he pasado horas divertidas.

Jerónimo García Gregorio

RESUMEN

Un ambiente de programación visual para graficar geometrías simples (líneas, rectángulos y círculos) mediante transformaciones ortogonales en dos lenguajes matemáticos: Álgebra Geométrica y Álgebra Vectorial; fue diseñado para poder establecer una comparación de la eficiencia numérica entre ambos lenguajes. En el diseño del ambiente se utilizó C++, la biblioteca multiplataforma **Qt** y el ambiente se ejecuta en la plataforma UBUNTU. Las clases fundamentales correspondientes a transformaciones ortogonales fueron diseñadas e implementadas con base en las características principales de sistemas de Diseño Asistido por Computadora (CAD, por sus siglas en inglés) comerciales y no comerciales. La experimentación numérica mostró que el Álgebra Geométrica presenta algunas ventajas con respecto al Álgebra Vectorial.

Palabras clave: Ambiente de programación visual, Geometría simples, Álgebra Geométrica, Álgebra Vectorial, Transformaciones Ortogonales, CAD.

ABSTRACT

An integrated development environment to plot simple geometries (lines, rectangle and circles) by orthogonal transformations in two mathematic languages: Geometric Algebra and Vectorial Algebra, was designed to establish a comparison of numerical efficiency between both languages. In the environment's design, C++, the library multiplatform **Qt** was employed and the environment runs in UBUNTU's platform. The fundamental class which correspond to orthogonal transformations were designed and implemented by taking into account the commercial and non-commercial CAD (Computer Assisted Design) system's main characteristics. The numerical experimentation showed that Geometric Algebra has some advantages with respect to Vectorial Algebra.

Keywords: Simple Geometry, Geometric Algebra, Vectorial Algebra, Orthogonal Transformations, CAD.

INDICE

| | No. De página |
|---|---------------|
| Introducción..... | 5 |
| Capitulo 1. Rotaciones y reflexiones del espacio 3D..... | 8 |
| Álgebra Vectorial..... | 8 |
| Reflexión Simple..... | 13 |
| Rotación simple..... | 14 |
| Álgebra Geométrica..... | 16 |
| Reflexión Simple..... | 21 |
| Rotación simple..... | 22 |
| Capitulo 2. Clases fundamentales del ambiente de programación visual..... | 24 |
| Definición del modelo..... | 24 |
| La clase TransformaciónOrtogonal con Álgebra Vectorial..... | 25 |
| La clase Matriz | 27 |
| La clase TransformaciónOrtogonal con Álgebra Geométrica..... | 30 |
| La clase Multivector | 30 |
| Capitulo 3. Interacción de la interfaz gráfica con el usuario..... | 37 |
| Construcción de una geometría simple..... | 42 |
| Conclusiones..... | 48 |
| Bibliografía..... | 49 |
| Apéndice 1. Demostración del teorema de Cartan para 3D..... | 50 |
| Apéndice 2. Manual del usuario..... | 51 |
| Apéndice 3. Código fuente del PDPACAD9..... | 63 |

Introducción

El Programa de Desarrollo Profesional en Automatización (PDPA) se encuentra dentro del marco del convenio que existe entre la UAM-Azcapotzalco y Parker Haniffin-México. Entre las actividades que se realizan en el PDPA se encuentra la optimización numérica de operaciones realizadas con sistemas de Diseño y Manufactura asistida por computadora (CAD/CAM [11], por sus siglas en inglés). En donde se pretende realizar las transformaciones ortogonales (rotaciones y reflexiones) con base en Álgebra Geométrica ([2] y [3]) para obtener algoritmos numéricos para optimizar sistemas CAD/CAM. Por lo que, entre los objetivos del PDPA es la formación de ingenieros que apliquen y amplíen sus conocimientos en CAD/CAM.

Las transformaciones ortogonales son importantes para construir geometrías de piezas a maquinar mediante sistemas CAD/CAM. Estos sistemas normalmente funcionan con matrices ortogonales para rotar o reflejar la geometría. En otras palabras, el lenguaje de Álgebra Vectorial (AV, ver [1] y [4]) es usado para realizar estas transformaciones. En la última década el Álgebra Geométrica (AG) ha emergido como una alternativa al AV al proporcionar un lenguaje conceptual más comprensivo, unificado y con un sistema computacional poderoso tanto simbólico como numérico. Los métodos de AV dependen fuertemente del uso de coordenadas, no obstante han sido utilizados para diversas tareas de programación geométrica. El AG presenta una alternativa a las limitaciones de vocabulario de AV como lenguaje de bajo nivel y provee de un lenguaje de alto nivel, poderoso para manipular objetos geométricos.

En este proyecto se diseñó un ambiente de programación visual para graficar geometrías simples (líneas, rectángulos y círculos) mediante transformaciones ortogonales en ambos lenguajes matemáticos AV y AG con la finalidad de establecer una comparación de la eficiencia numérica entre ambos lenguajes, en la construcción de piezas a maquinar mediante sistemas CAD/CAM. Este ambiente de programación corresponde a un sistema CAD el cual incluye además de AV también al AG. Con este sistema CAD se puede construir geometrías simples, con base en transformaciones ortogonales, y compara numéricamente los algoritmos correspondientes. La interfaz gráfica del sistema CAD permite al usuario: construir, visualizar y manipular, en forma dinámica, las geometrías simples y analizar la eficiencia numérica ya sea con AV o AG. El diseño estuvo guiado por los siguientes objetivos ([5]-[10]):

Objetivo General: Elaborar un paquete computacional en C++ para graficar geometrías simples (líneas, rectángulos y círculos) mediante transformaciones ortogonales y comparar su eficiencia numérica entre los lenguajes de AV y AG.

Objetivos particulares: Determinar algoritmos numéricos más eficientes para realizar matrices ortogonales; determinar las transformaciones ortogonales con base en álgebra geométrica; determinar criterios de eficiencia numérica; diseñar los algoritmos para implementar eficientemente las transformaciones ortogonales en C++, realizar la comparación, análisis de tiempo, número de operaciones y cantidad de memoria utilizada con ambas implementaciones: matrices y álgebra geométrica. desarrollar la interfaz gráfica para diseñar y visualizar figuras geométricas mediante puntos, líneas, círculos, rotaciones y reflexiones; describir y desarrollar el sistema en C++ construido con los resultados obtenidos de eficiencia y comparación numérica

Como resultado del objetivo general un sistema fue CAD diseñado con la ventaja de incluir además AG el cual se ejecuta en la plataforma UBUNTU 8.10 y requiere para su ejecución y la biblioteca multiplataforma para diseño de interfaz **Qt3** [10].

El reporte de este proyecto consiste en la memoria técnica con la documentación en donde están registrados el análisis y resultados obtenidos en este proyecto. La memoria describe las clases fundamentales en C++ y el sistema CAD con código fuente incluido bien documentado, manual de

usuario del funcionamiento de la interfaz gráfica, manual de instalación del producto y las pruebas de eficiencia numéricas realizadas. Específicamente, el reporte está organizado de la siguiente manera.

En el capítulo 1 presentamos las notaciones, terminología y resultados referentes al espacio euclidiano de tres dimensiones (3D) para describir las transformaciones ortogonales en los lenguajes de Álgebra Vectorial y Álgebra Geométrica. En el Apéndice 1 se demuestra para el espacio 3D un teorema de Cartan para la descomposición de una transformación ortogonal mediante reflexiones especulares.

En el capítulo 2 se describen las clases fundamentales implementadas en C++ para construir geometrías simples mediante rotaciones y reflexiones simples (figuras geométricas construidas con base en puntos, líneas, rectángulos y círculos) en un ambiente de visualización gráfica (interfaz gráfica). Primero describimos el almacenamiento de una geometría simple y su interfaz gráfica. Después se presentan las clases para las rotaciones y reflexiones simples en ambos lenguajes AV y AG.

En el Capítulo 3 se describen, primero, las clases que integran la interacción de la interfaz gráfica con el usuario para visualizar las geometrías simples en tres dimensiones. Después, se desarrolla en detalle la construcción de una geometría simple y finalmente se presenta una comparación del desempeño numérico cuando se utiliza AV y AG.

El Apéndice 2 contiene el manual de usuario del sistema CAD y su instalación con el cual él tendrá a su disposición las herramientas necesarias para introducir puntos, líneas y círculos y podrá rotar y reflejar para construir una geometría simple, podrá manipular visualmente en forma dinámica la geometría y con la opción de seleccionar el método para rotar y reflejar ya sea con AV y AG; también, el usuario podrá ver los valores de eficiencia numérica de rotar y reflejar con AM y AG de acuerdo a dos criterios computacionales: número de operaciones realizadas y memoria utilizada. Finalmente, el apéndice 3 contiene el código fuente incluido documentado del sistema CAD.

Capítulo 1

Rotaciones y reflexiones con Álgebra Vectorial y Geometría del espacio 3D

En este capítulo presentamos las notaciones, terminología y resultados referentes al espacio euclidiano tridimensional (3D) con el objetivo de describir las transformaciones ortogonales en los lenguajes de Álgebra Vectorial y Álgebra Geométrica. Primero se presentan las transformaciones y matrices ortogonales con álgebra vectorial. Después presentamos las transformaciones con Álgebra Geométrica. La mayoría de los resultados de Álgebra Vectorial son presentados sin demostración pero pueden ser encontrados en [1], asimismo los de álgebra geométrica en [2]. En la tercera sección presentamos a modo de apéndice una demostración algorítmica de un teorema de Cartan con Álgebra Geométrica.

Como es usual 3D es el conjunto de elementos $\mathbf{x} = (x_1, x_2, x_3)$; x_i un número real $i = 1, 2, 3$ llamados vectores en donde hay una suma y multiplicación por escalares (espacio vectorial) dados por:

$$\begin{aligned}\mathbf{x} + \mathbf{y} &= (x_1 + y_1, x_2 + y_2, x_3 + y_3) \\ \lambda \mathbf{x} &= (\lambda x_1, \lambda x_2, \lambda x_3)\end{aligned}$$

para todo $\mathbf{x} = (x_1, x_2, x_3)$, $\mathbf{y} = (y_1, y_2, y_3)$ en 3D y λ un escalar. En el espacio 3D existen subespacios de 1 o 2 dimensiones los cuales corresponden a líneas y planos que pasan por el origen, respectivamente. Sean $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ y $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ bases del espacio 3D y $\mathbf{P} = [p_{ij}]$ ($i, j = 1, 2, 3$) la matriz correspondiente al cambio de base de $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ a $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$. Entonces,

$$\mathbf{f}_j = \sum_{i=1}^3 p_{ij} \mathbf{e}_i ; j = 1, 2, 3$$

Entonces, decimos que $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ y $\{\mathbf{f}_1, \mathbf{f}_2, \mathbf{f}_3\}$ determinan la misma orientación en 3D si el determinante de \mathbf{P} es positivo i.e. $|\mathbf{P}| > 0$. Existe una relación de equivalencia sobre las bases en 3D mediante una partición en dos clases disjuntas. Bases que pertenecen a la misma clase proporcionan la misma orientación y si pertenecen a diferentes clases proporcionan diferente orientación.

El espacio 3D está orientado, si las bases que pertenecen a una de las clases es denominada positiva; las pertenecientes a la otra clase se denominan negativas. Como el cambio de bases queda completamente determinado por las transformaciones lineales en 3D decimos que una transformación lineal \mathbf{T} preserva la orientación si $|\mathbf{T}| = |\mathbf{A}| > 0$ en donde \mathbf{A} es la representación matricial de \mathbf{T} respecto a cualquier base: $\mathbf{A} = [\mathbf{T}(\mathbf{e}_1), \mathbf{T}(\mathbf{e}_2), \mathbf{T}(\mathbf{e}_3)]$ para cualquier base $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$.

Álgebra vectorial.

En álgebra vectorial de 3D están definidos dos productos: **escalar** y **vectorial**. Veamos primero algunas de las propiedades del producto escalar o interno. Geométricamente este producto, que

denotaremos por \cdot , proporciona *la proyección perpendicular de un vector en otro vector*, también resulta ser una regla algebraica que relaciona escalares con vectores con propiedades bien conocidas [1].

El producto escalar entre dos vectores $\mathbf{x} = (x_1, x_2, x_3)$; $\mathbf{y} = (y_1, y_2, y_3)$ en 3D está dado por

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^3 x_i y_i \quad (1)$$

y la longitud o norma de un vector es: $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$. Un vector \mathbf{u} es unitario si: $\|\mathbf{u}\| = 1$. Sean \mathbf{x}, \mathbf{y} en 3D vectores no-nulos, el ángulo entre \mathbf{x}, \mathbf{y} es un ángulo θ tal que

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

en donde $\theta \in [0, 2\pi]$. Dos vectores no-nulos $\mathbf{x}, \mathbf{y} \in \mathbb{R}^3$ son ortogonales o perpendiculares, $\mathbf{x} \perp \mathbf{y}$, si $\mathbf{x} \cdot \mathbf{y} = 0$ o equivalentemente $\theta = \frac{\pi}{2}$.

Si W es un subespacio en 3D (i.e. una línea o un plano que pasa por el origen), entonces su complemento ortogonal W^\perp es el subespacio, similarmente un plano o una línea según sea el caso, dado por:

$$W^\perp = \{\mathbf{x} \cdot \mathbf{y} = 0 \text{ para toda } \mathbf{x}, \mathbf{y} \text{ en } 3D\}$$

Al usar la descomposición de 3D como suma directa:

$$3D = W \oplus W^\perp$$

cada vector \mathbf{x} en 3D admite una única representación:

$$\mathbf{x} = \mathbf{x}_W + \mathbf{x}_{W^\perp}$$

en donde $\mathbf{x}_W \in W$ y $\mathbf{x}_{W^\perp} \in W^\perp$ las cuales son las proyecciones ortogonales de \mathbf{x} sobre W y W^\perp respectivamente. La proyección ortogonal \mathbf{x}_W queda bien caracterizada como el vector en W cuya distancia a \mathbf{x} es mínima:

$$\|\mathbf{x} - \mathbf{x}_W\| = \min_{\mathbf{y} \in W} \|\mathbf{x} - \mathbf{y}\|$$

también, es el único vector $\mathbf{x}_W \in W$ tal que $\mathbf{x} - \mathbf{x}_W \perp W$.

Ahora, si $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ es una base ortonormal (ON) de \mathbb{R}^3 i.e. $\|\mathbf{e}_i\| = 1; i = 1, 2, 3$ y $\mathbf{e}_i \perp \mathbf{e}_j$ para $i \neq j$, entonces,

$$\mathbf{x}_W = \sum_{i=1}^3 (\mathbf{x} \cdot \mathbf{e}_i) \mathbf{e}_i$$

La transformación lineal del espacio 3D al espacio 3D dada por:

$$\mathbf{T}_W(\mathbf{x}) = \mathbf{x} - \mathbf{x}_W$$

se llama reflexión del espacio 3D sobre W . En particular, el caso que nos interesa en este trabajo es cuando W será el complemento ortogonal de un vector no-nulo \mathbf{x} i.e. $W = \mathbf{x}^\perp$ el cual es geoméricamente un plano que pasa por el origen del espacio 3D. En este caso tenemos W es el subespacio generado por el vector \mathbf{x} y cualquier vector \mathbf{y} en 3D admite una única descomposición ortogonal:

$$\mathbf{y} = \mathbf{y}_\parallel + \mathbf{y}_\perp \quad (2)$$

en donde \mathbf{y}_\parallel pertenece al subespacio generado por \mathbf{x} y \mathbf{y}_\perp pertenece al plano \mathbf{x}^\perp -perpendicular a \mathbf{x} - y son las proyecciones ortogonales de \mathbf{y} sobre el subespacio generado por \mathbf{x} -proyección paralela- y \mathbf{x}^\perp -proyección perpendicular, respectivamente. Denotaremos las proyecciones por \mathbf{y}_\parallel y \mathbf{y}_\perp sin escribir explícitamente \mathbf{x} cuando se entienda en el contexto en que se está usando. Así

$$\mathbf{y}_\parallel = \lambda \mathbf{x} \text{ y } \mathbf{y}_\perp = \mathbf{y} - \mathbf{y}_\parallel$$

$$\mathbf{y}_\perp = (\mathbf{y} \cdot \mathbf{u}_1) \mathbf{u}_1 + (\mathbf{y} \cdot \mathbf{u}_2) \mathbf{u}_2$$

en donde $\{\mathbf{u}_1, \mathbf{u}_2\}$ es una base ON del plano \mathbf{x}^\perp y $\lambda = \frac{\mathbf{y} \cdot \mathbf{x}}{\|\mathbf{x}\|^2}$. Entonces, por definición la **reflexión simple** \mathbf{R}_x -reflexión a través del plano \mathbf{x}^\perp - está dada por (Figura1)

$$\mathbf{R}_x(\mathbf{y}) = -\mathbf{y}_\parallel + \mathbf{y}_\perp$$

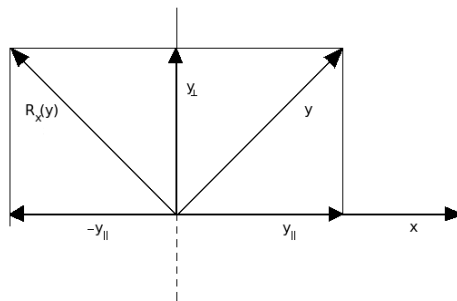


Figura 1: Reflexión simple (i.e. reflexión a través del plano \mathbf{x}^\perp)

Las reflexiones son un caso particular de las transformaciones lineales ortogonales en 3D. Una transformación lineal \mathbf{T} es ortogonal si satisface:

$$\mathbf{T}\mathbf{x} \cdot \mathbf{T}\mathbf{y} = \mathbf{x} \cdot \mathbf{y}$$

Equivalentemente, si \mathbf{A} es la matriz que representa a la transformación ortogonal \mathbf{T} respecto a una base. Entonces

$$\mathbf{T} \text{ es ortogonal} \Leftrightarrow \mathbf{A}'\mathbf{A} = \mathbf{A}\mathbf{A}' = \mathbf{I}_3$$

en donde \mathbf{A}' es la traspuesta de la matriz \mathbf{A} e \mathbf{I}_3 es la matriz identidad. Una matriz \mathbf{A} tal que $\mathbf{A}'\mathbf{A} = \mathbf{A}\mathbf{A}' = \mathbf{I}_3$ se llama matriz ortogonal.

Con el siguiente teorema, debido a Cartan [3], quedan caracterizadas todas las transformaciones ortogonales en 3D:

TEOREMA. *Toda transformación ortogonal en 3D es la composición de a lo más tres reflexiones simples.*

Por definición una transformación ortogonal es una **rotación simple** si es composición de dos reflexiones simples en los otros dos casos son una **reflexión** o reflexión simple. Además, una rotación tiene una representación matricial con determinante 1 y una reflexión con determinante -1 .

Con el teorema anterior quedan establecidas todas las transformaciones ortogonales en 3D. Una demostración de este teorema la presentaremos en el apéndice de la sección 3.

Como una rotación simple se obtiene con base en dos reflexiones simples, tenemos que determinar el eje de rotación y su ángulo. Para esto recurrimos a otro producto, además del escalar, el denominado producto cruz o vectorial entre dos vectores. Para definirlo consideremos la base canónica en 3D:

$$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$$

El producto vectorial entre dos vectores $\mathbf{x} = (x_1, x_2, x_3)$, $\mathbf{y} = (y_1, y_2, y_3)$ en 3D se obtiene de calcular simbólicamente el siguiente determinante:

$$\mathbf{x} \times \mathbf{y} = \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix} \quad (3)$$

Algunas de las propiedades del producto vectorial son:

1. $\mathbf{x} \times \mathbf{y} = -\mathbf{y} \times \mathbf{x}$
2. $\mathbf{x} \times \mathbf{y} \perp \mathbf{x}$ y $\mathbf{x} \times \mathbf{y} \perp \mathbf{y}$
3. $\mathbf{x} \times \mathbf{y} = \mathbf{0} \Leftrightarrow \mathbf{x} \parallel \mathbf{y}$
4. Si $\mathbf{x}, \mathbf{y} \neq \mathbf{0} \Rightarrow \|\mathbf{x} \times \mathbf{y}\| = \|\mathbf{x}\| \|\mathbf{y}\| \text{sen}\theta$; siendo θ el ángulo entre los vectores \mathbf{x}, \mathbf{y} .
5. $(\mathbf{x} \times \mathbf{y}) \cdot \mathbf{z} = \mathbf{x} \cdot (\mathbf{y} \times \mathbf{z}) = \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}$
6. $(\mathbf{y} \times \mathbf{x}) \times \mathbf{z} = (\mathbf{x} \cdot \mathbf{y})\mathbf{z} - (\mathbf{x} \cdot \mathbf{z})\mathbf{y}$ -triple producto vectorial.

En álgebra vectorial, las transformaciones ortogonales, pueden encontrarse mediante las proyecciones paralela y perpendicular de un vector y pueden ser escritas en términos de los productos escalar y vectorial [4]. En efecto, hemos considerado, en la ecuación (2), a y_{\parallel}^x , y_{\perp}^x como las proyecciones paralela y perpendicular del vector $y \neq 0$ sobre un vector fijo $x \neq 0$ (ver Figura 2), y si θ es el ángulo entre x e y , entonces,

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$

$$\mathbf{x} \times \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \sin \theta$$

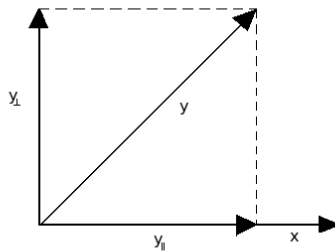


Figura 2. Proyección paralela y perpendicular de y sobre x.

De donde podemos inferir que si $\mathbf{a}, \mathbf{b} \neq 0$ son vectores arbitrarios, entonces:

$$\mathbf{a} \cdot \mathbf{y}_{\perp}^{\mathbf{a}} = \|\mathbf{a}\| \|\mathbf{y}_{\perp}^{\mathbf{a}}\| \cos \frac{\pi}{2} = 0$$

$$\|\mathbf{b} \times \mathbf{y}_{\parallel}^{\mathbf{b}}\| = \|\mathbf{b}\| \|\mathbf{y}_{\parallel}^{\mathbf{b}}\| \sin \pi = 0$$

ya que los productos escalar y vectorial son distributivos respecto a la suma:

$$\mathbf{a} \cdot \mathbf{y} = \mathbf{a} \cdot (\mathbf{y}_{\parallel}^{\mathbf{a}} + \mathbf{y}_{\perp}^{\mathbf{a}}) = \mathbf{a} \cdot \mathbf{y}_{\parallel}^{\mathbf{a}}$$

$$\mathbf{b} \times \mathbf{y} = \mathbf{b} \times (\mathbf{y}_{\parallel}^{\mathbf{b}} + \mathbf{y}_{\perp}^{\mathbf{b}}) = \mathbf{b} \times \mathbf{y}_{\perp}^{\mathbf{b}}$$

Así los productos $\mathbf{a} \cdot \mathbf{y}_{\parallel}^{\mathbf{a}}$ y $\mathbf{b} \times \mathbf{y}_{\perp}^{\mathbf{b}}$ contienen toda la **información** acerca de la proyección paralela y perpendicular de y a lo largo de los vectores \mathbf{a} y \mathbf{b} . Por lo tanto, si conocemos los valores de

$$\lambda = \mathbf{a} \cdot \mathbf{y}_{\parallel}^{\mathbf{a}}$$

y

$$\mathbf{c} = \mathbf{b} \times \mathbf{y}_{\perp}^{\mathbf{b}}$$

esto se convierte en plantear el siguiente problema: resolver las siguientes ecuaciones lineales [4]:

$$\mathbf{a} \cdot \mathbf{y} = \lambda \tag{4}$$

$$\mathbf{b} \times \mathbf{y} = \mathbf{c} \quad (5)$$

en donde \mathbf{a} , \mathbf{b} y \mathbf{c} son vectores fijos y λ es un escalar. Si los vectores \mathbf{a} , \mathbf{b} no son ortogonales la solución de las ecuaciones (4 y 5) es única: En el otro caso la solución no es única. En [4] se encuentra una discusión detallada de la unicidad. Supongamos, entonces, que los vectores \mathbf{a} , \mathbf{b} no son ortogonales así tendremos siempre una solución única.

Para resolver las ecuaciones (4 y 5) procedemos como sigue. Aplicamos la identidad, propiedad 6 correspondiente a el triple producto vectorial:

$$(\mathbf{b} \times \mathbf{y}) \times \mathbf{a} = (\mathbf{a} \cdot \mathbf{b})\mathbf{y} - (\mathbf{a} \cdot \mathbf{y})\mathbf{b}$$

Es importante agrupar los términos, ya que el producto cruz no es asociativo. Multiplicando mediante el producto vectorial por la izquierda por \mathbf{a} en ambos lados de la segunda ecuación (5):

$$\begin{aligned} (\mathbf{b} \times \mathbf{y}) \times \mathbf{a} &= \mathbf{c} \times \mathbf{a} \\ (\mathbf{a} \cdot \mathbf{b})\mathbf{y} - (\mathbf{a} \cdot \mathbf{y})\mathbf{b} &= \mathbf{c} \times \mathbf{a} \end{aligned}$$

y debido a la ecuación(4):

$$(\mathbf{a} \cdot \mathbf{b})\mathbf{y} - \lambda\mathbf{b} = \mathbf{c} \times \mathbf{a}$$

Ahora, despejando de la última ecuación a \mathbf{y} obtenemos la solución buscada:

$$\mathbf{y} = \frac{\lambda\mathbf{b} + \mathbf{c} \times \mathbf{a}}{\mathbf{a} \cdot \mathbf{b}} \quad (6)$$

Claramente si sustituimos este valor de \mathbf{y} en las ecuaciones(4 y 5) comprobamos que estas ecuaciones se satisfacen.

En particular, si $\mathbf{b} = \mathbf{a}$ la ecuación (6) se simplifica a:

$$\mathbf{y} = \frac{\lambda\mathbf{a} + \mathbf{c} \times \mathbf{a}}{\mathbf{a} \cdot \mathbf{a}} \quad (7)$$

aún más si \mathbf{a} es un vector unitario:

$$\mathbf{y} = \lambda\mathbf{a} + \mathbf{c} \times \mathbf{a} \quad (8)$$

Con la síntesis obtenida con las ecuaciones (4 y 5) y su respectiva solución ecuación (7 y 8) podemos obtener expresiones para la reflexión o rotación simples a partir del vector y plano de reflexión o de su eje de rotación y su ángulo, respectivamente. Esto lo mostramos a continuación.

Reflexión Simple

Como hemos visto la reflexión simple consiste en reflejar un vector \mathbf{v} a través de un plano cuya normal es un vector unitario \mathbf{n} . El vector reflejado \mathbf{v}^* tendrá la componente paralela a \mathbf{n} con signo menos, mientras que la componente perpendicular permanece invariante (ver ecuación (2)) como se muestra en la Figura 3.

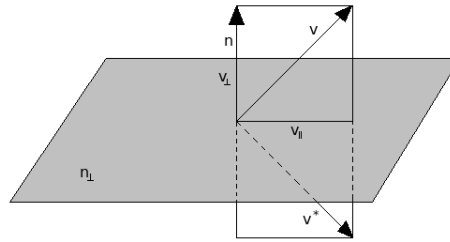


Figura 3. Reflexión del vector v en el plano n^\perp con vector normal n .

Como n^\perp es un plano fijo con vector unitario normal n . Entonces, las ecuaciones (4 y 5) son:

$$\mathbf{n} \cdot \mathbf{v}^* = \mathbf{n} \cdot \mathbf{v}$$

$$\mathbf{n} \times \mathbf{v}^* = -\mathbf{n} \times \mathbf{v}$$

y tienen como solución (sustituir en la ecuación (cuatro): $\mathbf{b} = \mathbf{a} = \mathbf{n}$, $\lambda = -\mathbf{n} \cdot \mathbf{v}$ y $\mathbf{c} = \mathbf{n} \times \mathbf{v}$):

$$\mathbf{v}^* = (\mathbf{n} \times \mathbf{v}) \times \mathbf{n} - (\mathbf{n} \cdot \mathbf{v})\mathbf{n}$$

$$\mathbf{v}^* = \mathbf{v} - 2(\mathbf{n} \cdot \mathbf{v})\mathbf{n}$$

y si el vector normal n no es unitario:

$$\mathbf{v}^* = \mathbf{v} - \frac{2(\mathbf{n} \cdot \mathbf{v})}{\|\mathbf{n}\|^2} \mathbf{n} \quad (9)$$

Rotación simple.

Dados un vector n unitario y un ángulo θ , hay que encontrar el vector rotado v^* de un vector arbitrario v con eje de rotación n y con un ángulo θ . Para esto basta solamente rotar la componente del vector v , v_\perp , que se encuentra en el plano cuya normal es n (Figura 4).

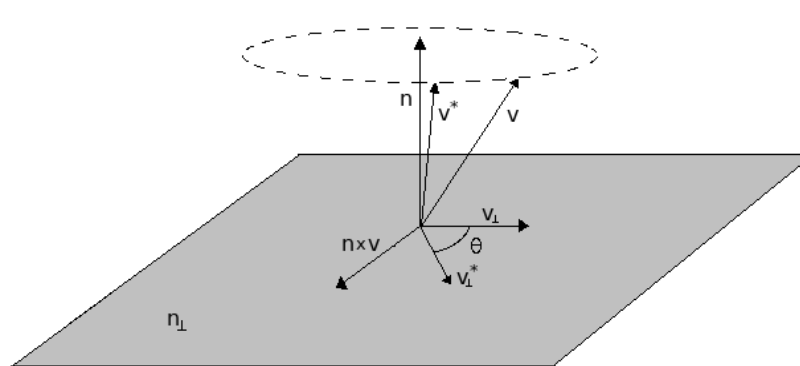


Figura 4. Rotación del vector v en v^* con eje de rotación n y con un ángulo θ .

Entonces las ecuaciones (4 y 5) son:

$$\mathbf{n} \cdot \mathbf{v}^* = \mathbf{n} \cdot \mathbf{v}$$

$$\mathbf{n} \times \mathbf{v}^* = \mathbf{n} \times (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta)$$

La segunda ecuación se cumple debido a que, ver Figura 5:

$$\mathbf{v}_{\perp}^* = (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}_{\perp}) \operatorname{sen} \theta)$$

así

$$\mathbf{n} \times \mathbf{v}_{\perp}^* = \mathbf{n} \times (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}_{\perp}) \operatorname{sen} \theta)$$

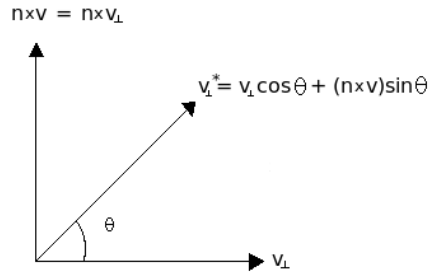


Figura 5: Vista de la rotación en el plano perpendicular al eje de rotación

Pero

$$\mathbf{n} \times \mathbf{v}_{\perp}^* = \mathbf{n} \times \mathbf{v}_{\perp}$$

$$\mathbf{n} \times \mathbf{v}_{\perp} = \mathbf{n} \times \mathbf{v}$$

entonces,

$$\mathbf{n} \times \mathbf{v}^* = \mathbf{n} \times (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta)$$

Por lo tanto, la solución es (sustituir en la ecuación (cuatro): $\mathbf{b} = \mathbf{a} = \mathbf{n}$, $\lambda = \mathbf{n} \cdot \mathbf{v}$ y

$\mathbf{c} = \mathbf{n} \times (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta)$):

$$\mathbf{v}^* = (\mathbf{n} \times (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta)) \times \mathbf{n} + (\mathbf{n} \cdot \mathbf{v}) \mathbf{n}$$

para poder simplificar esta ecuación sea $\mathbf{u} = \mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta$:

$$\mathbf{v}^* = (\mathbf{n} \times \mathbf{u}) \times \mathbf{n} + (\mathbf{n} \cdot \mathbf{v}) \mathbf{n}$$

por la propiedad 6 del producto vectorial tenemos, ya que \mathbf{n} es unitario:

$$\mathbf{v}^* = \mathbf{u} - (\mathbf{n} \cdot \mathbf{u}) \mathbf{n} + (\mathbf{n} \cdot \mathbf{v}) \mathbf{n}$$

y sustituyendo el valor de \mathbf{u} , y como $\mathbf{n} \cdot (\mathbf{n} \times \mathbf{v}) = 0$ (propiedad 2 del producto vectorial):

$$\mathbf{v}^* = \mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta - (\mathbf{n} \cdot (\mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta)) \mathbf{n} + (\mathbf{n} \cdot \mathbf{v}) \mathbf{n}$$

$$\mathbf{v}^* = \mathbf{v} \cos \theta + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta - (\mathbf{n} \cdot \mathbf{v} \cos \theta) \mathbf{n} + (\mathbf{n} \cdot \mathbf{v}) \mathbf{n}$$

y al factorizarla obtenemos:

$$\mathbf{v}^* = \mathbf{v} \cos \theta + (1 - \cos \theta)(\mathbf{n} \cdot \mathbf{v})\mathbf{n} + (\mathbf{n} \times \mathbf{v}) \operatorname{sen} \theta \quad (10)$$

que es la conocida **fórmula de Rodrigues** para una rotación.

La representación matricial de la reflexión y rotación simples se obtienen de evaluar:

$$\mathbf{v}^* = \mathbf{T}(\mathbf{v})$$

en la base canónica en 3D:

$$\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$$

es decir son las matrices:

$$[\mathbf{T}(\mathbf{e}_1), \mathbf{T}(\mathbf{e}_2), \mathbf{T}(\mathbf{e}_3)]$$

para la reflexión simple:

$$\begin{bmatrix} \frac{-n_1^2 + n_2^2 + n_3^2}{\|\mathbf{n}\|^2} & -\frac{2n_1n_2}{\|\mathbf{n}\|^2} & -\frac{2n_1n_3}{\|\mathbf{n}\|^2} \\ -\frac{2n_1n_2}{\|\mathbf{n}\|^2} & \frac{n_1^2 - n_2^2 + n_3^2}{\|\mathbf{n}\|^2} & -\frac{2n_2n_3}{\|\mathbf{n}\|^2} \\ -\frac{2n_1n_3}{\|\mathbf{n}\|^2} & -\frac{2n_2n_3}{\|\mathbf{n}\|^2} & \frac{n_1^2 + n_2^2 - n_3^2}{\|\mathbf{n}\|^2} \end{bmatrix} \quad (11)$$

y para la rotación simple:

$$\begin{bmatrix} \frac{n_1^2 + (n_2^2 + n_3^2) \cos \theta}{\|\mathbf{n}\|^2} & \frac{n_1n_2(1 - \cos \theta) + \|\mathbf{n}\|n_3 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} & \frac{n_1n_3(1 - \cos \theta) - \|\mathbf{n}\|n_2 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} \\ \frac{n_1n_2(1 - \cos \theta) - \|\mathbf{n}\|n_3 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} & \frac{n_2^2 + (n_1^2 + n_3^2) \cos \theta}{\|\mathbf{n}\|^2} & \frac{n_2n_3(1 - \cos \theta) + \|\mathbf{n}\|n_1 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} \\ \frac{n_1n_3(1 - \cos \theta) + \|\mathbf{n}\|n_2 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} & \frac{n_2n_3(1 - \cos \theta) - \|\mathbf{n}\|n_1 \operatorname{sen} \theta}{\|\mathbf{n}\|^2} & \frac{n_3^2 + (n_1^2 + n_2^2) \cos \theta}{\|\mathbf{n}\|^2} \end{bmatrix} \quad (12)$$

Algebra geométrica en 3D

Sea $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ base en 3D, entonces, podemos construir una Álgebra lineal asociativa de dimensión maximal que contenga a los números reales y al espacio 3D y que satisfaga las siguientes reglas de multiplicación ([2] y [3]):

$$\begin{aligned} \mathbf{e}_i^2 &= 1; \quad i = 1, 2, 3 \\ \mathbf{e}_i \mathbf{e}_j + \mathbf{e}_j \mathbf{e}_i &= 0; \quad i \neq j \end{aligned} \quad (13)$$

Por lo que tenemos la siguiente definición:

DEFINICIÓN: El álgebra asociativa y distributiva generada por el espacio euclidiano en 3D con las reglas de multiplicación dadas por (13) en donde $\{e_1, e_2, e_3\}$ es una base en 3D, es llamada álgebra de Clifford universal o álgebra geométrica del espacio 3D y es denotada por GA_3 .

El espacio vectorial GA_3 tiene dimensión $2^3 = 8$ con una base dada por

$$\{1, e_1, e_2, e_3, e_1e_2, e_2e_3, e_3e_1, e_1e_2e_3\}$$

Los elementos $e_{12} = e_1e_2, e_{23} = e_2e_3, e_{31} = e_3e_1$ son llamados bivectores, geoméricamente son segmentos de plano orientados xy, yz, xz (ver Figura 6), y los denotamos por $e_{ij}; i \neq j$. Al elemento $e_1e_2e_3$ lo denotamos por e_{123} y se le llama trivector o pseudoescalar, geoméricamente es un segmento de espacio orientado (Figura 6) y nos proporciona la orientación positiva.

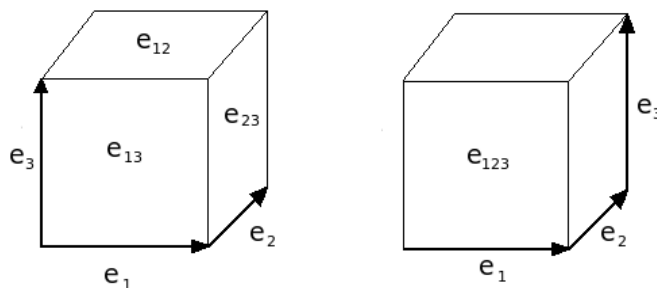


Figura 6. Segmentos de plano orientados e_{12}, e_{23}, e_{31} y segmento de espacio orientado e_{123} .

Cualquier elemento (multivector) $M \in \mathbb{R}_3$ es una combinación lineal y puede escribirse como:

$$M = a_0 + \sum_{i=1}^3 a_i e_i + \sum_{\substack{i,j=1 \\ j \neq i}}^3 a_{ij} e_{ij} + a_{123} e_{123} \quad (14)$$

El espacio generado por $\{1\}$ es el espacio de escalares y queda identificado con los números reales y sus elementos serán denotados por a, b, c, \dots . El subespacio generado por $\{e_1, e_2, e_3\}$ es el espacio 3D y por (13) resulta ser una base ON de 3D y sus elementos los denotamos por a, b, c, \dots . El subespacio generado por $\{e_{12}, e_{23}, e_{31}\}$ es llamado el espacio de bivectores y sus combinaciones lineales -elementos- los denotamos por A, B, \dots . Finalmente el subespacio generado por $\{e_{123}\}$ es llamado espacio de trivectores o pseudoescalares de la forma: αe_{123} , donde α pertenece a los reales.

PROPOSICIÓN. Si a, b pertenecen espacio 3D entonces $a^2 \geq 0$ y $ab + ba$ es un número real. Aún más

$$a^2 = a \cdot a \quad (15)$$

$$\frac{ab + ba}{2} = a \cdot b \quad (16)$$

en donde \cdot corresponde al producto escalar en 3D.

DEMOSTRACIÓN. Es suficiente escribir \mathbf{a} y \mathbf{b} en términos de la base $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ y usar la ecuación (15).

Esta proposición establece la relación entre el producto escalar en 3D y el así llamado **producto geométrico** del AG_3 . De hecho constituye la parte simétrica de este producto. La parte antisimétrica del producto geométrico se emplea para definir el producto exterior $\mathbf{a} \wedge \mathbf{b}$ de dos vectores \mathbf{a}, \mathbf{b} en 3D:

$$\mathbf{a} \wedge \mathbf{b} = \frac{\mathbf{ab} - \mathbf{ba}}{2} \quad (17)$$

Con esta definición el producto geométrico de dos vectores \mathbf{a}, \mathbf{b} en 3D puede escribirse como:

$$\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b} \quad (18)$$

De (13) y (15) obtenemos:

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= \mathbf{b} \cdot \mathbf{a} \\ \mathbf{a} \wedge \mathbf{b} &= -\mathbf{b} \wedge \mathbf{a} \end{aligned}$$

En 3D todo vector no-nulo tiene inverso. En efecto, sea \mathbf{a} en los reales, $\mathbf{a} \neq 0$, el inverso está dado por:

$$\mathbf{a}^{-1} = \frac{\mathbf{a}}{\mathbf{a}^2} = \frac{\mathbf{a}}{\|\mathbf{a}\|^2} \quad (19)$$

El producto geométrico (18) relaciona propiedades algebraicas con propiedades geométricas. En particular, tenemos las siguientes propiedades fundamentales:

$$\begin{aligned} \mathbf{ab} = \mathbf{ba} &\Leftrightarrow \mathbf{a} \parallel \mathbf{b} \Leftrightarrow \mathbf{a} \wedge \mathbf{b} = 0 \\ \mathbf{ab} = -\mathbf{ba} &\Leftrightarrow \mathbf{a} \perp \mathbf{b} \Leftrightarrow \mathbf{a} \cdot \mathbf{b} = 0 \end{aligned}$$

Por lo que, *vectores que son paralelos conmutan y vectores ortogonales o perpendiculares anticommutan mediante el producto geométrico.*

Ahora, si $\mathbf{a} = \sum_{i=1}^3 a_i \mathbf{e}_i$, $\mathbf{b} = \sum_{i=1}^3 b_i \mathbf{e}_i$, entonces,

$$\mathbf{a} \wedge \mathbf{b} = \begin{vmatrix} \mathbf{e}_{12} & \mathbf{e}_{23} & \mathbf{e}_{31} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \quad (20)$$

y como puede verse sus componentes son las mismas que las correspondientes a las del producto

vectorial (3).

Además, los tres bivectores básicos satisfacen:

$$\mathbf{e}_{ij}^2 = -1; i \neq j$$

y al multiplicarlos por un vector básico lo transforma en otro vector básico. Por ejemplo,

$$\mathbf{e}_1 \mathbf{e}_{12} = \mathbf{e}_2$$

El producto geométrico de vectores se extiende linealmente a todos los elementos del álgebra en 3D por lo que podemos formar expresiones como \mathbf{aB} en donde \mathbf{B} es un bivector arbitrario. Pero como \mathbf{e}_{123} es un trivector, entonces, el resultado de multiplicar puede contener a vectores y bivectores.

Para ver la propiedades del producto \mathbf{aB} descomponemos a \mathbf{a} mediante la ecuación (2) (ver Figura 7):

$$\mathbf{a} = \mathbf{a}_{\parallel} + \mathbf{a}_{\perp} = \mathbf{a}_{\parallel} + \mathbf{a}_{\perp}$$

entonces,

$$\mathbf{aB} = (\mathbf{a}_{\parallel} + \mathbf{a}_{\perp})\mathbf{B}$$

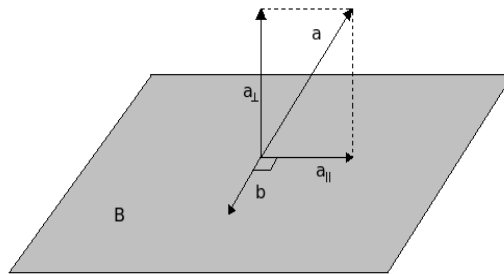


Figura 7: El vector \mathbf{a} se descompone, respecto a \mathbf{B} , como suma de dos vectores: uno perteneciendo al plano y el otro perpendicular.

Podemos suponer sin pérdida de generalidad que:

$$\mathbf{B} = \mathbf{a}_{\parallel} \wedge \mathbf{b}$$

con \mathbf{b} un vector ortogonal a \mathbf{a}_{\parallel} . Por lo que

$$\mathbf{a}_{\parallel} \mathbf{B} = \mathbf{a}_{\parallel} (\mathbf{a}_{\parallel} \wedge \mathbf{b}) = \mathbf{a}_{\parallel} (\mathbf{a}_{\parallel} \mathbf{b}) = \mathbf{a}_{\parallel}^2 \mathbf{b} \tag{21}$$

es decir, $\mathbf{a}_{\parallel} \mathbf{B}$ es un vector en la dirección de \mathbf{b} . Por otra parte,

$$\mathbf{a}_{\perp} \mathbf{B} = \mathbf{a}_{\perp} (\mathbf{a}_{\parallel} \wedge \mathbf{b}) = \mathbf{a}_{\perp} \mathbf{a}_{\parallel} \mathbf{b} \tag{22}$$

que es el producto geométrico de tres vectores ortogonales, es decir, $\mathbf{a} \perp \mathbf{B}$ es un trivector. Por lo tanto el producto geométrico de un vector y un bivector proporciona dos términos: un vector y un trivector.

$$\mathbf{aB} = \mathbf{a} \cdot \mathbf{B} + \mathbf{a} \wedge \mathbf{B} \quad (23)$$

en donde $\mathbf{a} \cdot \mathbf{B} = \mathbf{a}_{\parallel} \mathbf{B}$ proyecta al vector \mathbf{a} sobre el plano, lo rota $\frac{\pi}{2}$ y después lo dilata la magnitud de \mathbf{B} (ver ecuación(21) y Figura 7). Además

$$\mathbf{a} \cdot \mathbf{B} = \mathbf{a}_{\parallel}^2 \mathbf{b} = -(\mathbf{a}_{\parallel} \mathbf{b}) \mathbf{a}_{\parallel} = -\mathbf{B} \mathbf{a}_{\parallel} = -\mathbf{B} \cdot \mathbf{a}$$

así el producto entre vector y bivector es antisimétrica. Entonces, definimos el producto interno de un vector y un bivector mediante:

$$\mathbf{a} \cdot \mathbf{B} = \frac{\mathbf{aB} - \mathbf{Ba}}{2}$$

para ver que este producto siempre es un vector podemos realizar el siguiente cálculo:

$$\begin{aligned} \mathbf{a} \cdot (\mathbf{b} \wedge \mathbf{c}) &= \frac{\mathbf{a}(\mathbf{b} \wedge \mathbf{c}) - (\mathbf{b} \wedge \mathbf{c})\mathbf{a}}{2} = \frac{\mathbf{a}(\mathbf{bc} - \mathbf{cb}) - (\mathbf{bc} - \mathbf{cb})\mathbf{a}}{4} \\ &= \frac{(\mathbf{ab})\mathbf{c} - (\mathbf{ac})\mathbf{b} - (\mathbf{bc} - \mathbf{cb})\mathbf{a}}{4} \\ &= \frac{(2\mathbf{a} \cdot \mathbf{b} - \mathbf{ba})\mathbf{c} - (2\mathbf{a} \cdot \mathbf{c} - \mathbf{ca})\mathbf{b} - (\mathbf{bc} - \mathbf{cb})\mathbf{a}}{4} \\ &= \frac{(2\mathbf{a} \cdot \mathbf{b})\mathbf{c} - \mathbf{b}(\mathbf{ac}) - (2\mathbf{a} \cdot \mathbf{c})\mathbf{b} + \mathbf{c}(\mathbf{ab}) - (\mathbf{bc} - \mathbf{cb})\mathbf{a}}{4} \\ &= \frac{(2\mathbf{a} \cdot \mathbf{b})\mathbf{c} - (2\mathbf{a} \cdot \mathbf{c})\mathbf{b} + \mathbf{bca} - (2\mathbf{a} \cdot \mathbf{c})\mathbf{b} + (2\mathbf{a} \cdot \mathbf{b})\mathbf{c} + \mathbf{cba} - (\mathbf{bc} - \mathbf{cb})\mathbf{a}}{4} \\ &= (\mathbf{a} \cdot \mathbf{b})\mathbf{c} - (\mathbf{a} \cdot \mathbf{c})\mathbf{b} \end{aligned}$$

lo cual resulta ser un vector. Así, hemos obtenido la siguiente identidad:

$$\mathbf{a} \cdot (\mathbf{b} \wedge \mathbf{c}) = (\mathbf{a} \cdot \mathbf{b})\mathbf{c} - (\mathbf{a} \cdot \mathbf{c})\mathbf{b} \quad (24)$$

Ahora por la ecuación(22), $\mathbf{a} \wedge \mathbf{B}$ proyecta sobre la componente perpendicular al plano y proporciona un trivector. Este término es simétrico:

$$\mathbf{a} \wedge \mathbf{B} = \mathbf{a}_{\perp} \mathbf{a}_{\parallel} \mathbf{b} = \mathbf{a}_{\parallel} \mathbf{b} \mathbf{a}_{\perp} = \mathbf{B} \wedge \mathbf{a}$$

por lo que el producto exterior de un vector y un bivector queda definido por:

$$\mathbf{a} \wedge \mathbf{B} = \frac{\mathbf{aB} + \mathbf{Ba}}{2}$$

El pseudoescalar \mathbf{e}_{123} es el único trivector unitario en 3D que proporciona orientación positiva y es tal

$$(\mathbf{a} \wedge \mathbf{b}) \cdot (\mathbf{c} \wedge \mathbf{d}) = (\mathbf{b} \cdot \mathbf{c})(\mathbf{a} \cdot \mathbf{d}) - (\mathbf{b} \cdot \mathbf{d})(\mathbf{a} \cdot \mathbf{c})$$

es decir, el producto escalar de dos bivectores es un escalar. En efecto, al aplicar la linealidad de la proyección y las ecuaciones (18), (23) y (24):

$$\begin{aligned} (\mathbf{a} \wedge \mathbf{b}) \cdot (\mathbf{c} \wedge \mathbf{d}) &= \langle (\mathbf{a} \wedge \mathbf{b})(\mathbf{c} \wedge \mathbf{d}) \rangle_0 \\ &= \langle (\mathbf{a}\mathbf{b} - \mathbf{a} \cdot \mathbf{b})(\mathbf{c} \wedge \mathbf{d}) \rangle_0 \\ &= \langle \mathbf{a}(\mathbf{b}(\mathbf{c} \wedge \mathbf{d})) \rangle_0 \\ &= \langle \mathbf{a}(\mathbf{b} \cdot (\mathbf{c} \wedge \mathbf{d}) + \mathbf{b} \wedge (\mathbf{c} \wedge \mathbf{d})) \rangle_0 \\ &= \langle \mathbf{a}(\mathbf{b} \cdot (\mathbf{c} \wedge \mathbf{d})) \rangle_0 \\ &= \langle \mathbf{a} \cdot (\mathbf{b} \cdot (\mathbf{c} \wedge \mathbf{d})) + \mathbf{a} \wedge (\mathbf{b} \cdot (\mathbf{c} \wedge \mathbf{d})) \rangle_0 \\ &= \langle \mathbf{a} \cdot (\mathbf{b} \cdot (\mathbf{c} \wedge \mathbf{d})) \rangle_0 \\ &= \mathbf{a} \cdot ((\mathbf{b} \cdot \mathbf{c})\mathbf{d} - (\mathbf{b} \cdot \mathbf{d})\mathbf{c}) \\ &= (\mathbf{a} \cdot \mathbf{d})(\mathbf{b} \cdot \mathbf{c}) - (\mathbf{a} \cdot \mathbf{c})(\mathbf{b} \cdot \mathbf{d}) \end{aligned}$$

en donde hemos aplicado varias veces $\langle \mathbf{M} \rangle_0 = 0$ cuando \mathbf{M} es un vector, bivector o trivector.

La reflexión y rotación simples resultan ser elementos del álgebra geométrica AG_3 . A continuación presentamos estos hechos.

Reflexión simple.

Dados dos vectores \mathbf{v} y \mathbf{n} , entonces, al vector \mathbf{v} podemos descomponerlo mediante la ecuación (desorto) en:

$$\mathbf{v} = \mathbf{v}_{\parallel} + \mathbf{v}_{\perp}$$

en donde \mathbf{v}_{\parallel} es paralelo a \mathbf{n} y \mathbf{v}_{\perp} es ortogonal a \mathbf{n} , así,

$$\mathbf{n}\mathbf{v}_{\parallel} = \mathbf{v}_{\parallel}\mathbf{n}$$

$$\mathbf{n}\mathbf{v}_{\perp} = -\mathbf{v}_{\perp}\mathbf{n}$$

Entonces, por la ecuación (gcinco)

$$\mathbf{n}\mathbf{v} = \mathbf{n}(\mathbf{v}_{\parallel} + \mathbf{v}_{\perp}) = (\mathbf{v}_{\parallel} - \mathbf{v}_{\perp})\mathbf{n}$$

al multiplicar por $-\mathbf{n}^{-1}$ a ambos lados de la ecuación:

$$-\mathbf{n}\mathbf{v}\mathbf{n}^{-1} = (-\mathbf{v}_{\parallel} + \mathbf{v}_{\perp}) \quad (27)$$

la cual corresponde a la ecuación (reflex). Por lo tanto el vector \mathbf{v}^* reflejado, mediante una reflexión simple a través del plano cuya normal es el vector \mathbf{n} , del vector \mathbf{v} resulta ser (ver Figura 3):

$$\mathbf{v}^* = -\mathbf{nv}\mathbf{n}^{-1}$$

También esto lo podemos obtener de la siguiente forma alternativa:

$$\begin{aligned} -\mathbf{nv}\mathbf{n}^{-1} &= -(\mathbf{nv})\mathbf{n}^{-1} = -(\mathbf{n} \cdot \mathbf{v} + \mathbf{n} \wedge \mathbf{v})\mathbf{n}^{-1} \\ &= -\left(\mathbf{n} \cdot \mathbf{v} + \frac{\mathbf{nv} - \mathbf{vn}}{2}\right)\mathbf{n}^{-1} = -(\mathbf{n} \cdot \mathbf{v})\mathbf{n}^{-1} - \left(\frac{\mathbf{nv} - \mathbf{vn}}{2}\right)\mathbf{n}^{-1}. \end{aligned}$$

Al resolver para $-\mathbf{nv}\mathbf{n}^{-1}$ y usar la ecuación (19) del inverso de un vector obtenemos:

$$-\mathbf{nv}\mathbf{n}^{-1} = \mathbf{v} - \frac{2(\mathbf{n} \cdot \mathbf{v})}{\|\mathbf{n}\|^2} \mathbf{n}$$

que corresponde a la ecuación(reflejado). Si \mathbf{n} es un vector unitario la reflexión simple se escribe como:

$$-\mathbf{nv}\mathbf{n}$$

Rotación simple.

Como sabemos que una rotación simple es el producto de dos reflexiones simples. De la Figura 8 puede verse que el resultado de dos reflexiones simples sucesivas es rotar a través de un ángulo 2θ en el plano $\mathbf{m}_1 \wedge \mathbf{m}_2$, en donde $\mathbf{m}_1 \cdot \mathbf{m}_2 = \cos\theta$ con $\mathbf{m}_1, \mathbf{m}_2$ vectores unitarios. Si \mathbf{n} es un vector unitario que corresponde al eje de rotación y θ es el ángulo de rotación, entonces al multiplicar a \mathbf{n} por el pseudoescalar e_{123} obtenemos un bivector \mathbf{B} :

$$\mathbf{B} = e_{123}\mathbf{n}$$

el cual corresponde al plano de rotación. Por lo tanto, $\mathbf{B} = \mathbf{m}_1 \wedge \mathbf{m}_2$.

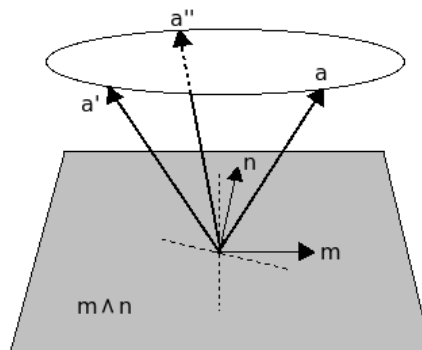


Figura 8: Rotación simple obtenida de dos reflexiones simples.

La primera refleja el vector \mathbf{v} en el plano cuya normal es \mathbf{m}_2 y después se obtiene \mathbf{v}^* reflejando en el plano cuya normal es \mathbf{m}_1

Entonces, la rotación simple buscada es:

$$\mathbf{v}^* = \mathbf{m}_1 \mathbf{m}_2 \mathbf{v} \mathbf{m}_2 \mathbf{m}_1$$

En efecto, veamos el caso en que los vectores \mathbf{m}_1 , \mathbf{m}_2 sean unitarios, su producto geométrico es:

$$\mathbf{m}_1 \mathbf{m}_2 = \cos \theta + \mathbf{m}_1 \wedge \mathbf{m}_2 \quad (29)$$

y por la identidad(27)

$$\begin{aligned} |\mathbf{m}_1 \wedge \mathbf{m}_2|^2 &= \mathbf{m}_1 \cdot (\mathbf{m}_2 \cos \theta - \mathbf{m}_1) \\ &= \cos^2 \theta - 1 \\ &= -\text{sen}^2 \theta \end{aligned}$$

por lo que si definimos el bivector unitario:

$$\bar{\mathbf{B}} = \frac{\mathbf{m}_2 \wedge \mathbf{m}_1}{\text{sen} \theta}; \quad \bar{\mathbf{B}}^2 = -1$$

la orientación de este bivector coincide con la de la rotación (Figura 4). Así la ecuación (29) puede escribirse como:

$$\mathbf{m}_1 \mathbf{m}_2 = \cos \theta - \bar{\mathbf{B}} \text{sen} \theta$$

es decir,

$$\mathbf{m}_1 \mathbf{m}_2 = e^{-\bar{\mathbf{B}} \theta}$$

pero la rotación fue a través de un ángulo 2θ .

Por lo tanto la fórmula final ajustada es:

$$\mathbf{v}^* = e^{-\frac{\bar{\mathbf{B}} \theta}{2}} \mathbf{v} e^{\frac{\bar{\mathbf{B}} \theta}{2}} \quad (30)$$

la cual describe una rotación simple a través de un ángulo θ en el plano $\bar{\mathbf{B}}$, con orientación especificada por el bivector $\bar{\mathbf{B}}$ tal que su eje de rotación es: $\mathbf{e}_{123} \bar{\mathbf{B}} = \mathbf{n}$.

Con la ecuación (rotaE) es fácil ver que la matriz de rotación:

$$\left[e^{-\frac{\bar{\mathbf{B}} \theta}{2}} \mathbf{e}_1 e^{\frac{\bar{\mathbf{B}} \theta}{2}}, e^{-\frac{\bar{\mathbf{B}} \theta}{2}} \mathbf{e}_2 e^{\frac{\bar{\mathbf{B}} \theta}{2}}, e^{-\frac{\bar{\mathbf{B}} \theta}{2}} \mathbf{e}_3 e^{\frac{\bar{\mathbf{B}} \theta}{2}} \right]$$

es idéntica a la matriz dada por la ecuación(12).

Capítulo 2

Clases fundamentales del ambiente de programación visual

En este capítulo se describen las clases fundamentales implementados en C++ [5] para construir geometrías simples mediante rotaciones y reflexiones simples (figuras geométricas construidas con base en líneas, rectángulos y círculos) en un ambiente de visualización gráfica; de aquí en adelante denominada interfaz gráfica. Primero se describe el almacenamiento de una geometría simple y su interfaz gráfica. Después se describen las clases para las rotaciones y reflexiones simples en ambos lenguajes Álgebras Vectoriales y Geométricas.

A continuación presentamos algunos preliminares sobre clases:

Las clases representan un conjunto de objetos con propiedades y comportamientos comunes, las clases representan cualquier tipo de “objeto”, desde software, hardware o simplemente conceptos. Los atributos son datos asociados a los elementos que toman valor al hacer instancias objetos de una clase. Los métodos son funciones o procesos propios de los objetos de una clase.

Las clases se presentan mediante diagramas siguiendo el estilo del Lenguaje Unificado de Modelado [6] (Unified Modeling Language, UML por sus siglas en inglés), v. gr. un rectángulo con compartimientos internos, en el compartimiento superior esta el nombre de la clase, en el compartimiento de enmedio estarán especificados los atributos y por ultimo en el compartimiento inferior estarán los métodos de la clase.

Implementación de la interfaz gráfica y almacenamiento de una geometría simple.

La interfaz gráfica requiere del uso de una estructura-objeto dinámico [5] para almacenar los datos (rectángulo, líneas y círculos) que serán introducidos por el usuario. La estructura debe ser dinámica para que el usuario tenga flexibilidad para construir geometrías simples con diferentes tamaños y formas; de este modo la interfaz gráfica realizará la asignación de memoria en tiempo real mediante un arreglo con dimensión variable. La dimensión del arreglo aumentará o disminuirá en función de las acciones realizadas por el usuario.

Definición del modelo

Las geometrías simples quedan definidas mediante dos conjuntos: uno de puntos o vectores tridimensionales y otro de características con solamente un atributo: color de la geometría. El primer conjunto consiste en una clase **Punto3D** la cual representa un punto o vector en el espacio tridimensional (3D) y provee un conjunto de estos objetos para obtener una lista que defina los límites de la geometría. En la Figura 1 se muestra **Punto3D** y la clase paleta de colores **PaletaColores** que define el color de la geometría seleccionado por el usuario.

Para definir las operaciones aplicables a la geometría, primero debemos contar con una estructura que construya la geometría, esto lo hacemos con base en una lista ligada [5] a **Punto3D**, la construcción se realiza a partir de la entrada de datos del usuario con ayuda del ratón o teclado. El usuario previamente puede seleccionar algún color para la geometría, esto se realiza con la paleta de colores incluida (ver Capítulo 3 y Apéndice 2 para más detalles).

El modelo quedó estructurado mediante estas clases de objetos como lo muestra la Figura 1. A continuación describimos en más detalle cada una de estas clases.

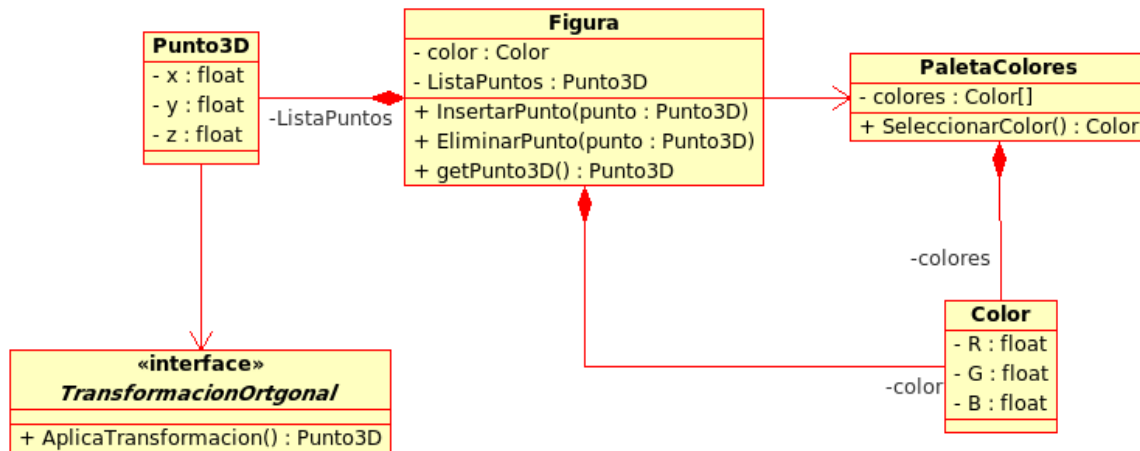


Figura 1. Las clases que constituyen el Modelo.

Si para construir la geometría el usuario utiliza una rotación o reflexión simples; las cuales fueron descritas en el Capítulo 1 (ecuaciones (9 y 10) y (27 y 30)), la clase **TransformacionOrtogonal** es la que corresponde a la realización de esas rotaciones o reflexiones simples. Como la rotación o reflexión simples se aplican a vectores o puntos 3D de la clase **Punto3D**, entonces, obtenemos como resultado el conjunto completo de puntos correspondiente a la geometría transformada.

Así, el modelo consiste de las siguientes cinco clases:

1. **Punto3D** proporciona un objeto el cual almacenara las coordenadas (x, y, z) de la colección de puntos inicial que conforman la geometría simple.
2. **Figura** proporciona el conjunto de puntos de la geometría inicial con opción a colorearse mediante las dos siguientes clases:
3. **PaletaColores**, con la cual se puede seleccionar el color de la geometría inicial hasta la geometría final, con base en:
4. **Color**, la cual contiene tres parámetros de colores primarios los cuales varían para obtener una gama de colores amplia.

Como entre los objetivos de este trabajo es analizar la eficiencia numérica, mediante dos métodos, para construir geometrías, entonces, la clase

5. **TransformacionOrtogonal** la cual realiza rotaciones y reflexiones simples para construir la geometría, con respecto.

Tiene una doble función: construir esta geometría con base en Álgebra vectorial y Álgebra Geométrica. Por lo que esta clase funciona tanto con métodos matriciales como multivectoriales, respectivamente. El usuario tiene que seleccionar el método con el cual construirá la geometría simple (ver Apéndice 2)

La clase TransformaciónOrtogonal con Álgebra Vectorial.

Los elementos principales del Álgebra Vectorial para realizar transformaciones ortogonales son: vectores, producto escalar o vectorial y operaciones con matrices, una discusión detallada se encuentra en el Capítulo 1, Estos elementos son los que implementamos en esta clase para llevar a cabo estas transformaciones.

A continuación describimos el modelo correspondiente a la clase **TransformaciónOrtogonal**. Este modelo consiste en cinco clases las cuales contienen los algoritmos necesarios, para rotar y reflejar geometrías simples con métodos de Álgebra Lineal y son las siguientes (ver Figura 2):

1. **matriz**, Construye las interrelaciones con las cuatro clases restantes.
2. **reflexion**, Realiza las operaciones necesarias para llevar a cabo una reflexión simple (ecuación (9)).
3. **rotacion**, Realiza las operaciones necesarias para llevar a cabo la rotación simple (ecuación (10)).
4. **productoPunto**, Encargada de hacer las operaciones para calcular el producto punto entre dos vectores.
5. **productoCruz**, Encargada de hacer las operaciones para calcular el producto Cruz entre dos vectores.

La Figura 2 ilustra las relaciones existentes entre estas las clases e indica cómo es la relación de estas clases entre sí. A continuación describimos en más detalle cada una de estas clases.

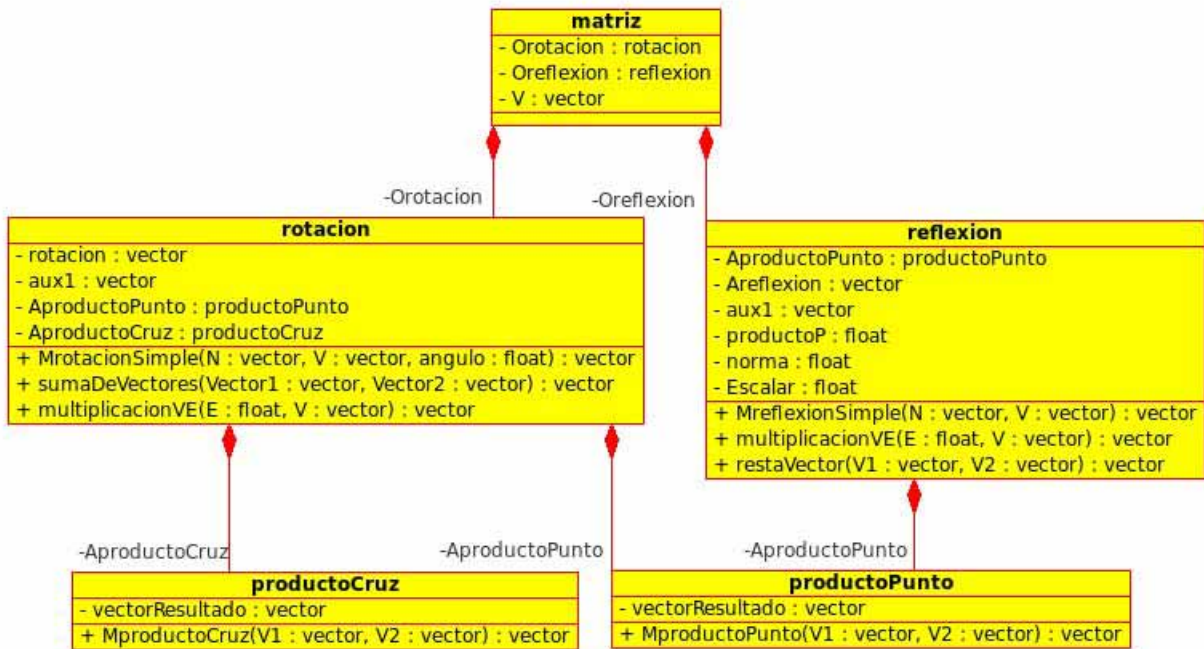


Figura 2. Modelo de la clase TransformaciónOrtogonal.

La clase matriz

Esta clase inicia con la interacción entre las otras clases, esta clase puede interpretarse como la puerta de comunicación para poder comunicarnos con las otras clases restantes. Con **matriz** se establece la conexión con las clases **rotacion** y **reflexion**; esto es debido a que en la clase **matriz** es donde se toma la decisión del tipo de transformación ortogonal que se usará y son **reflexion** o **rotacion** las encargadas de llevar a cabo esta transformación, Claramente la clase **reflexion** lleva a cabo una reflexión simple y la clase **rotacion** lleva a cabo una rotación simple.

Como ilustra la Figura 2, la clase **matriz** solamente tiene atributos sin incluir métodos; ya que es con clase la llama a las clases **reflexión** y la **rotación**. Por lo tanto en esta clase no se hacen operaciones, únicamente tiene tres atributos uno es un objeto de la clase **reflexion** denominado *Oreflexion* y otro de la clase **rotacion** denominado *Orotacion*. Para evitar confusión con objetos, métodos y clases que tienen un nombre idéntico; a los objetos se le colocará una O al inicio, análogamente a los atributos que no crean un objeto una A y a los métodos una M. Por ejemplo, el objeto *Oreflexión* creado en **matriz** hace uso del método correspondiente *MreflexionSimple*. El tercer atributo corresponde a la variable tipo **vector** la cual nos representa un vector; **vector** es una estructura que contiene los tres vectores básicos e_1 , e_2 y e_3 (como variables tipo float).

Se emplea una estructura para manejar vectores en lugar de un arreglo 3D por compatibilidad con el método multivectorial el cual usa esta estructura como clase; ya que en este caso hay que considerar el inverso de un vector. Por lo que, esta variable **vector** además de almacenar vectores también funciona como vector normal al plano de reflexión, eje de rotación o para almacenar el resultado cuando se obtiene un vector de productos geométricos entre vectores.

La clase productoPunto

El producto punto de dos vectores se realiza con esta clase y corresponde a realizar la sumatoria de los productos de las componentes de los dos vectores (ver Capítulo 1). El resultado es un escalar el cual corresponde al atributo *escalar* de la clase **productoPunto**, es de tipo float y será almacenado como atributo. El método que hace esta operación es *MproductoPunto* el cual admite dos parámetros de tipo vector y proporciona un escalar.

La clase reflexión

Esta clase la cual es llamada desde la clase **matriz** construye el objeto *Oreflexion* (ver Figura 2) y es el que permite acceder a los métodos de la clase **reflexion**. Es en esta clase en donde se realizan las operaciones para obtener una reflexión simple (ver ecuación (9)), como son: los cálculos correspondientes al producto punto, norma y operaciones entre matrices y vectores.

La clase tiene cinco atributos, los cuales hemos llamado: *AproductoPunto*, *Areflexion*, *aux1*, *norma* y *productoP*. Con el primero de los atributos se construye el objeto *productoPunto* de la clase **productoPunto** para poder acceder al método *MproductoPunto*. Este método realiza el producto punto entre dos vectores vía componentes (ver Capítulo 1). Con el segundo atributo se tiene un vector llamado *Areflexion* en donde se almacena el vector obtenido como resultado de reflejar un vector con respecto al vector normal del plano de reflexión, el tercero corresponde a un vector auxiliar *aux1* que se utiliza para almacenar resultados parciales, el cuarto *norma* almacena la normal al plano de reflexión y en la variable float *productoP* se almacena el escalar que resulta del producto punto entre la norma y el vector a reflejar.

En lo referente a los métodos, tenemos primero a *MreflexionSimple* en el cual se realizan las operaciones de la reflexión simple indicadas en la ecuación (9) y se auxilia de dos métodos. El

método da como resultado el vector reflejado, si los parámetros de entrada son dos vectores que corresponden al vector normal al plano de reflexión y a un vector arbitrario el cual será reflejado. Un método auxiliar es *multiplicacionVE* ya que solamente multiplica un escalar con un vector: se obtiene como resultado un vector, si las parámetros de entrada: son un variable float escalar y vector de tipo **vector**. Por último, *restaVector* un método auxiliar ya que solamente proporciona un tercer vector resultado de la resta de dos vectores.

La forma en que se desarrolla la ecuación (9) para obtener el vector reflejado con los métodos y atributos es como sigue: el método *MreflexionSimple* es el principal puesto que con el invocamos a los métodos restantes *multiplicacionVE* y *restaVector* para realizar todas las operaciones e invocamos a la clase **productoPunto**. En efecto, primero invocamos a *MproductoPunto* con el cual se realiza el producto punto entre los vectores normal al plano de reflexión y un vector arbitrario, el resultado es un primer escalar que se almacena en el atributo *productoP*. Segundo, se calcula la norma al cuadrado del vector normal y se almacena en el atributo *norma*. Tercero, con el método *multiplicacionVE* se obtiene, el cual es almacenado en *aux1*, un vector que resulta de multiplicar el primer escalar por el vector normal al plano de reflexión y después dividirlo por lo que hay en *norma*. Finalmente, con *restaVector* se realiza la resta entre el vector arbitrario y *aux1*, y obtenemos vector reflejado el cual se almacena en *vectorResultado*.

La Figura 3 ilustra la clase **Reflexion** mediante un diagrama secuencial. A continuación presentamos en forma resumida este diagrama:

1. La clase **matriz** inicia la comunicación con las clases **Reflexion** mediante *MreflexionSimple* al enviar como parámetros al vector normal al plano de reflexión y a un vector arbitrario.
2. De la clase **Reflexion** también se invoca a *MproductoPunto* de la clase **productoPunto**. El método admite al vector arbitrario y al vector normal al plano de reflexión y realiza el producto punto entre estos vectores.
3. Se efectúa la multiplicación del vector normal al plano de reflexión por el escalar con *multiplicacionVE*, proporciona como resultado el vector el cual se almacena en *aux1*.
4. Por último se obtiene la resta entre el vector arbitrario y el vector *aux1* con *restaVector*.

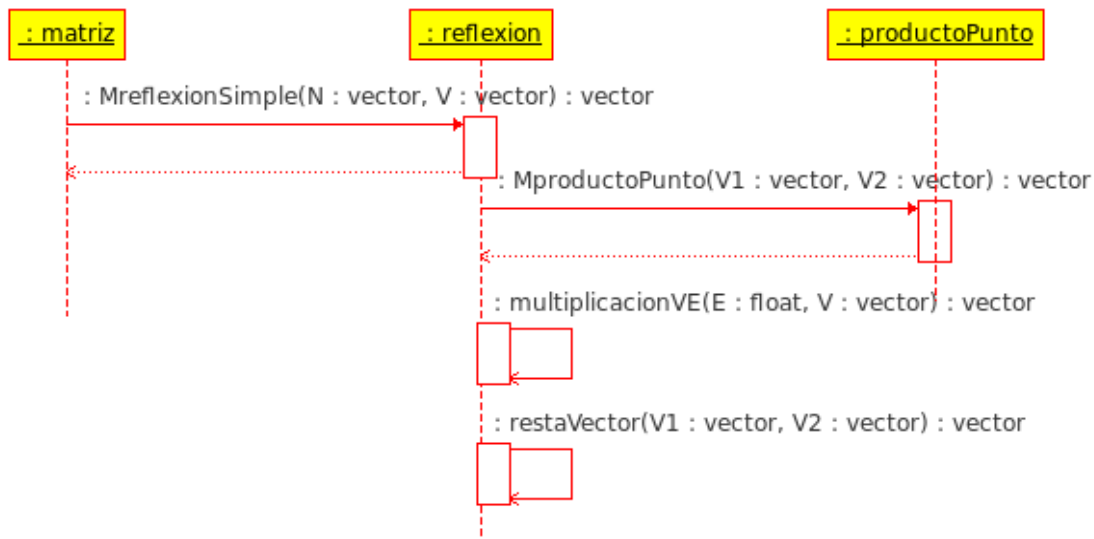


Figura 3. Diagrama secuencial de la clase Reflexión.

La clase productoCruz

El producto cruz o vectorial entre dos vectores, ecuación (3), proporciona como resultado un vector ortogonal a ambos. La clase **productoCruz** es la encargada de realizar esta operación.

La clase rotacion

Análogamente, a la clase la **reflexión**, esta clase es invocada desde la clase **matriz** al construir el objeto *rotacion* con el cual se lleva a cabo una rotación simple.

Esencialmente en esta clase se realizan los cálculos indicados en la ecuación (10) que corresponde a una rotación simple (**fórmula de Rodriguez**). En lo que sigue, se describen los atributos y métodos con los que cuenta esta clase. La clase cuenta con cuatro atributos: *rotacion*, *aux1*, *AproductoPunto* y *AproductoCruz*:

AproductoPunto y *MproductoPunto* corresponden al mismo atributo y método de la clase **reflexión**. El vector *aux1* almacenara vectores y es auxiliar en las operaciones intermedias de la rotación simple. El vector *rotacion* almacena el resultado (vector rotado). Por último tenemos al objeto *AproductoCruz* construido para acceder a la clase **productoCruz** mediante *MproductoCruz*. El método *MrotacionSimple* de la clase **rotacion** invoca a los métodos *multiplicacionVE*, realiza la misma acción que en la clase **reflexión**, *sumaDeVectores*, realiza la suma entre dos vectores. Así *MrotacionSimple* proporciona como resultado al vector rotado el cual queda almacenado en *rotacion* con parámetros de entrada el eje de rotación, el vector que se va a rotar y el ángulo de rotación.

En resumen, *MrotacionSimple* funciona con dos vectores y *angulo*, realiza los productos punto y cruz entre los dos vectores el cual se almacena en *aux1*. Después se multiplica por el coseno del ángulo, por el seno del ángulo al producto cruz y por uno menos el coseno del ángulo multiplicado por el producto punto de los vectores y por el eje de rotación. Termina, cuando se *sumaDeVectores*, suma los tres vectores resultantes, y el resultado se almacena en *rotacion*.

La Figura 4 ilustra la clase **Rotacion** mediante un diagrama secuencial. A continuación presentamos en forma resumida este diagrama:

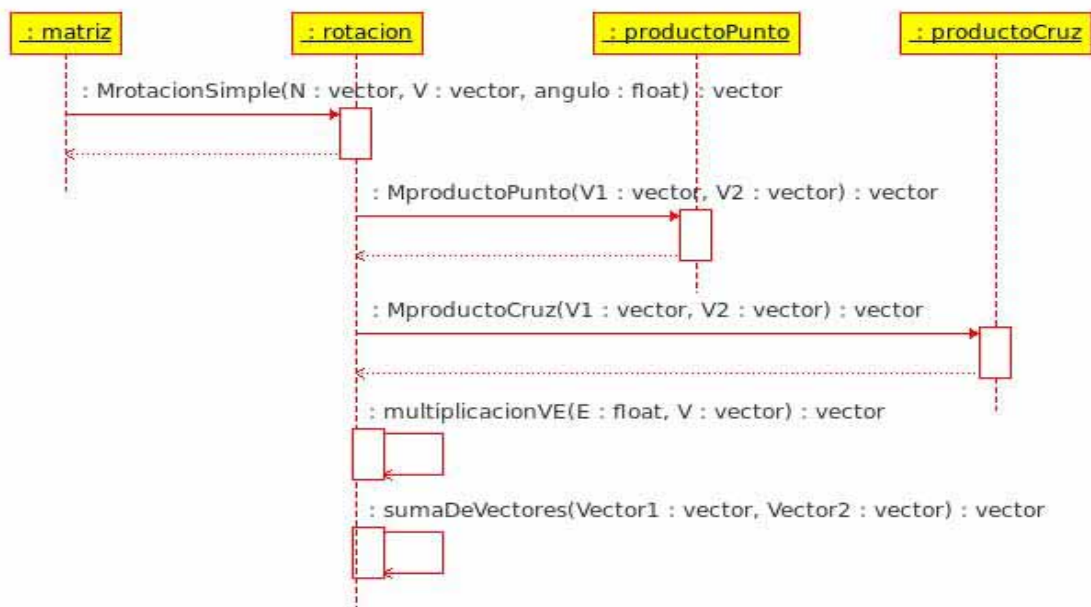


Figura 4. Diagrama secuencial de la clase Rotacion.

1. 1. La clase **matriz** inicia la comunicación con las clases **Rotacion** mediante *MrotacionSimple*, con parámetros de entrada el eje de rotación, el vector que se va a rotar y el ángulo de rotación.
2. Dentro de la clase **Rotacion** se realiza *MproductoPunto* de la clase **productoPunto**.
3. Se efectúa la multiplicación con *MproductoCruz* de **productoCruz**, el resultado queda almacenado en *aux1*.
4. Después se usa tres veces *multiplicacionVE* (ecuación (10)).
5. Por último, se utiliza tres veces *sumaDevectores*, para obtener así al vector rotado.

La clase **TransformaciónOrtogonal** con Álgebra Geométrica

Los elementos principales del Álgebra Geométrica para realizar transformaciones ortogonales son: multivectores y las operaciones en esta álgebra, una discusión detallada se encuentra el Capítulo 1, Estos elementos son los que implementamos en esta clase para llevar a cabo estas transformaciones. A continuación, se describe la clase **TransformaciónOrtogonal** la cual corresponde a la aplicación transformaciones ortogonales, mediante álgebra geométrica, a elementos de la clase **Punto3D** (ecuaciones (28 y 30)). Para este fin se construyó un modelo para los elementos y operaciones fundamentales en esta álgebra para el espacio 3D. Como se describió en el Capítulo 1 la rotación y reflexión simples resultan ser elementos de esta álgebra.

Este modelo consiste en seis clases las cuales contienen los algoritmos necesarios, para rotar y reflejar geometrías simples con métodos de Álgebra Geométrica, y son las siguientes (ver Figura 5):

1. **Multivector**. Encargada de los multivectores 8D (ecuación (14)).
2. **ProductoInterior**. Realiza las operaciones necesarias del producto interior entre los objetos *multivector*.
3. **ProductoExterior**. Realiza las operaciones necesarias del producto exterior entre los objetos *multivector*.
4. **ProductoGeométrico** Realiza la suma entre **ProductoInterior** y **ProductoExterior** cuando multivector contiene un vector o un bivector (ecuaciones (18 y 14)).
5. **Reflexión**. Refleja **Punto3D** (o bivectores) a través de un plano de reflexión, dado por el usuario, mediante **ProductoGeometrico** (ecuación(28))
6. **Rotación**, Rota un objeto **Punto3D** (o bivectores) a través de un plano de rotación dado por el usuario mediante **ProductoGeometrico** (ecuación(30))

La clase **Multivector**

El objeto fundamental dentro del modelo es *multivector* ya que con en este objeto se aplican las operaciones correspondientes a la clase **TransformaciónOrtogonal**. En esta clase hay cuatro clases fundamentales, para más detalles ver Capítulo 1, las cuales son:

1. **Escalar**. almacena una componente que representa la parte 0-vectorial de *multivector*.
2. **Vector**, almacena tres componentes que representan la parte vectorial del *multivector*.
3. **Bivector**, almacena también tres componentes que representa la parte 2-vectorial del multivector.
4. **Trivector**, este objeto almacena una componente que representa la parte 3-vectorial del multivector.

Cada una de estas clases contendrá sus respectivos métodos ver Figura 5, los cuales son: *suma*, *resta*, *magnitud* e *inverso* (reverso en caso de *bivector* y *trivector*). De esta manera, *r* contendrá a cada una de estos métodos como se muestra en la Figura 5. También *Multivector* se encarga de invocar a las otras clases. Otras operaciones aplicables a *Multivector* son: **ProductoInterior**, **ProductoExterior** y **ProductoGeométrico**, que corresponden a clases las cuales están interrelacionadas con la clase **Multivector** y serán discutidas más adelante. La Figura 5 muestra estas interrelaciones.

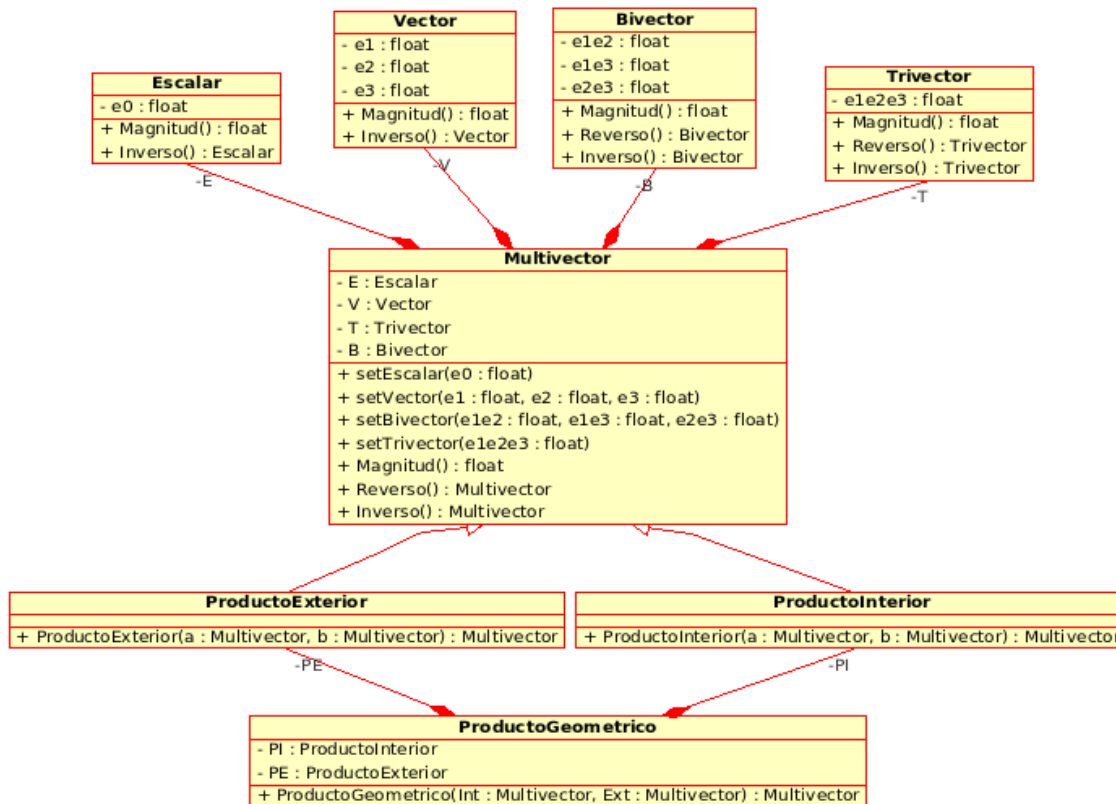


Figura 5. Diagrama del modelo de la clase **Multivector**.

Método *magnitud* de un multivector

Con la ecuación (27) del Capítulo 1 se encuentra el atributo *magnitud* de *Multivector* mediante la sumatoria de la magnitud de sus partes *r*-vectoriales. Con *magnitud* se obtiene la magnitud de cada uno de los *r*-vectores mediante la raíz cuadrada de la sumatoria de sus componentes elevados al cuadrado. Entonces, con *magnitud* de *Multivector* se obtiene cada una de las magnitudes de sus

diferentes componentes (pasos 1 a 4 de la Figura 6,) y después se realiza la sumatoria de todos (paso 5) para obtener el resultado.

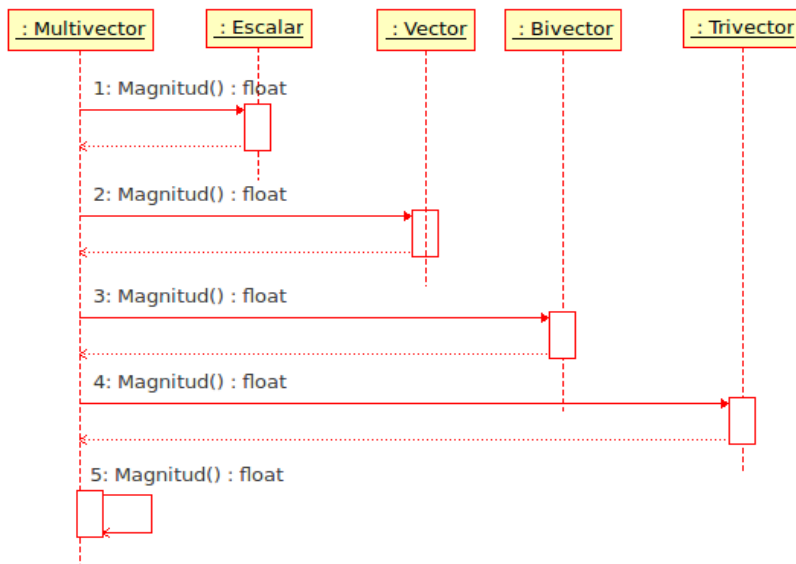


Figura 6. Diagrama secuencial de *magnitud*.

Método *Inverso* de un Multivector

La ecuación (19) corresponde a la fórmula para obtener el inverso (*Inverso*) de un vector: dividir el vector por *magnitud*, en donde está almacenada la *magnitud* del vector elevada al cuadrado. Para r-vectores de dimensión mayor se requiere *Reverso* (ecuación (26) capítulo 1). Con *Inverso* obtenemos el inverso de un vector, bivector o trivector. La Figura 7 muestra el diagrama secuencial de este método.

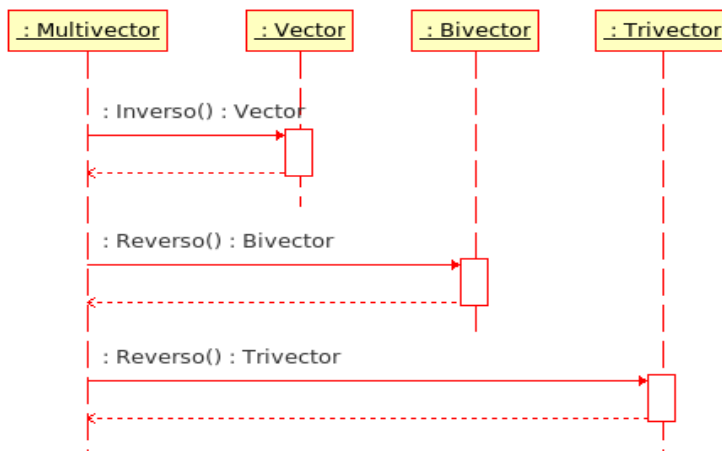


Figura 7. Diagrama secuencial de *Inverso*.

Las clases *ProductoInterior* y *ProductoExterior*.

Las clases **ProductoInterior** y **ProductoExterior** proporcionan como resultado un r-vector correspondiente cuando se operan a las diversas parejas de r-vectores en estas. Después de evaluar

los diferentes casos que pueden ocurrir para cada una de las diferentes parejas de r-vectores, se obtienen las tablas 1 y 2.

| R-vector1 | R-vector2 | R-vector resultado | Propiedad |
|-----------|-----------|--------------------|-----------------|
| Escalar | Escalar | 0 | |
| Escalar | Vector | 0 | |
| Escalar | Bivector | 0 | |
| Escalar | Trivector | 0 | |
| Vector | Vector | Escalar | Conmutativa |
| Vector | Bivector | Vector | Anticonmutativa |
| Vector | Trivector | Bivector | Conmutativa |
| Bivector | Bivector | Escalar | Conmutativa |
| Bivector | Trivector | Vector | Conmutativa |

Tabla 1. Resultados obtenidos para la clase ProductoInterior.

| R-vector1 | R-vector2 | R-vector resultado | Propiedad |
|-----------|-----------|--------------------|-----------------|
| Escalar | Escalar | Escalar | Conmutativa |
| Escalar | Vector | Vector | Conmutativa |
| Escalar | Bivector | Bivector | Conmutativa |
| Escalar | Trivector | Trivector | Conmutativa |
| Vector | Vector | Bivector | Anticonmutativa |
| Vector | Bivector | Trivector | Conmutativa |
| Vector | Trivector | --- | |
| Bivector | Bivector | --- | |
| Bivector | Trivector | --- | |

Tabla 1. Resultados obtenidos para la clase ProductoExterior.

En estas tablas se encuentran los resultados que fueron implementados. Claramente, las que están sin marcar no fueron consideradas.

La clase **ProductoGeométrico**

Con las ecuaciones (18 y 23) se obtiene el producto geométrico entre vectores y vector-bivector, Por lo que la clase **ProductoGeométrico** admite solamente vectores o vector-bivector. Para realizar este producto se invocan las clases **ProductoInterior** y **ProductoExterior** en forma secuencial (pasos 1 y 2 de la Figura 8), con dos *Multivector* se aplica *suma* para obtener el producto geométrico (paso 3 de la Figura 8).

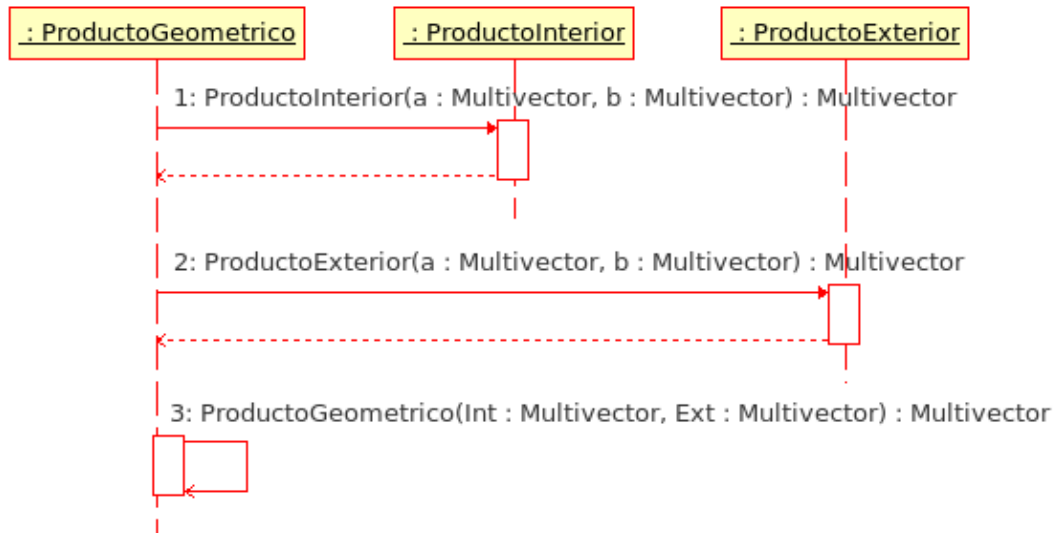


Figura 8. Diagrama secuencial de la clase Producto Geométrico.

La clase Reflexión

La ecuación (reflexion) corresponde a una reflexión simple: con **ProductoGeometrico** del vector normal y el vector a reflejar se obtiene un bivector y con **ProductoGeometrico** del bivector y el vector normal se obtiene el vector reflejado en el plano con ese vector normal, previamente se multiplica por menos. La clase **Reflexion** es la encargada de realizar lo anterior al invocar **Punto3D**.

La clase **Reflexion** realiza secuencialmente estas operaciones y con la clase **Figura** se obtiene parte o toda la geometría simple construida. A continuación se describe la secuencia de la clase **Reflexión**:

1. Admite el vector normal al plano y el vector. mediante **Vector**, a ser reflejado.
2. Con **Inverso** y **Vector** se obtiene el inverso del vector normal.
3. Se selecciona el primer vector de la lista de objetos **Punto3D** de la clase **Figura**.
4. Se selecciona otro vector y continúa con el siguiente y así sucesivamente hasta agotar la lista de vectores en **Figura**.
5. Se aplica **ProductoGeometrico** con cada **Punto3D** y el vector normal al plano, se almacena en *aux*.
6. Se aplica **ProductoGeometrico** a *aux* e **Inverso** y se inserta en **Figura** y así sucesivamente hasta obtener finalmente la geometría construida.

La Figura 9 ilustra el diagrama secuencial de esta clase.

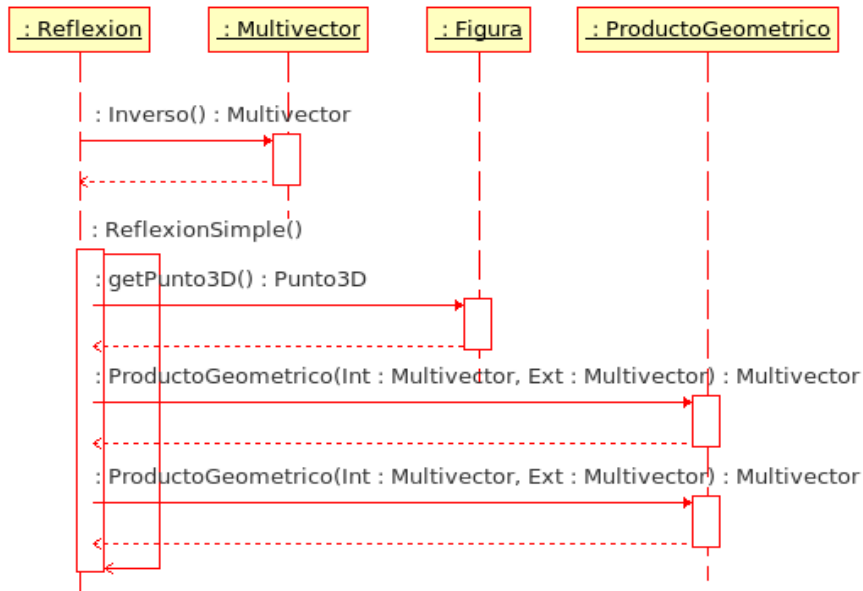


Figura 9. Diagrama secuencial de la clase Reflexión.

La clase Rotacion

La ecuación (30) corresponde a una rotación. En este caso solamente se necesita aplicar el producto Geométrico para obtener un bivector (con parte escalar y parte bivectorial) el cual se denomina rotor. La clase **Rotación** admite un conjunto de vectores de la clase **Punto3D** el cual representan la geometría de la clase **Figura** que será rotada con un ángulo y eje de rotación dados. La clase **Rotación** involucra un rotor (ecuación (29)) el cual el coseno del ángulo de rotación representa una escalar y la parte bivectorial es el seno de este ángulo multiplicado por el pseudoescalar del plano de rotación.

La clase **Rotacion** realiza secuencialmente estas operaciones y con la clase **Figura** se obtiene parte o toda la geometría simple construida. A continuación se describe la secuencia de la clase **Reflexión**:

1. Admite el eje y ángulo de rotación y el vector, mediante **Vector**, a rotar.
2. Con **Rotor** y **RotorInverso** se realiza la rotación.
3. Se selecciona el primer vector de la lista de objetos **Punto3D** de la clase **Figura**.
4. Se selecciona otro vector y continúa con el siguiente y así sucesivamente hasta agotar la lista de vectores en **Figura**.
5. Se aplica **ProductoGeometrico** entre **Rotor** y el vector a rotar, se almacena en *aux*.
7. Se aplica **ProductoGeometrico** entre *aux* y **RotorInverso** y se inserta en **Figura** y así sucesivamente hasta obtener finalmente la geometría construida.

La Figura 10 ilustra el diagrama secuencial de esta clase.

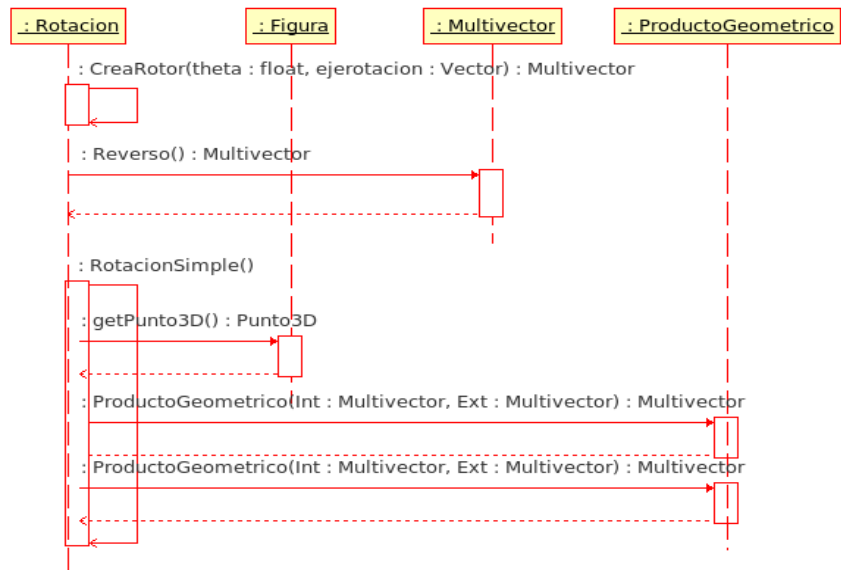


Figura 10. Diagrama secuencial de la clase Rotación.

Capítulo 3

Interacción de la interfaz gráfica con el usuario y construcción de una geometría simple.

En este capítulo se describe primero las clases que integran la interacción de la interfaz gráfica con el usuario para visualizar las geometrías simples en tres dimensiones. Después, se desarrolla detalladamente la construcción de una geometría simple y finalmente presentamos una comparación del desempeño numérico cuando se utiliza Álgebra Lineal y Álgebra Geométrica.

Interacción de la interfaz gráfica con el usuario

Para la interacción de la interfaz gráfica con el usuario realizamos la integración completa del sistema para su funcionamiento con la finalidad de construir geometrías simples y poder así determinar el número de operaciones realizadas en función de una u otra Álgebra.

Primeramente describimos las clases fundamentales del ambiente integrado; para después mostrar el funcionamiento de cada una de las partes que lo integran. A continuación, mostramos las clases y un diagrama que ilustra las interrelaciones entre estas clases (Diagrama 1).

6. **Pantalla.** Esta clase es la parte fundamental del ambiente pues contiene todas las herramientas que integran el sistema y permite la comunicación entre ellas, así como también mostrar el estado del sistema en la pantalla.
7. **Menu.** Esta clase permite desplegar en la pantalla un conjunto de opciones disponibles para el usuario. Internamente indica las tareas que desarrolla en el ambiente visual.
8. **Paleta.** La clase paleta despliega una ventana para que el usuario elija el color que usara para construir la geometría simple.
9. **Extruir.** Con esta clase se realiza la operación de extruir una geometría después de haberla seleccionado. Cuenta con opciones para elegir la dimensión y la dirección que desea el usuario realizar la extrusión.
10. **Operaciones.** Es una clase que despliega una ventana con el número de operaciones realizadas cuando se aplica una reflexión o rotación simples.
11. **Gibox.** Es una clase diseñada en OpenGL que corresponde a la ventana principal y es la que muestra las geometrías construidas por el usuario. En esta ventana es donde el usuario puede construir geometrías.
12. **Figura.** Esta clase contiene las geometrías que se encuentran en memoria. Para más detalles ver capítulo 2.
13. **TransformacionOrtogonal.** Es la clase que controla a las reflexiones y rotaciones simples.

La Figura 1 muestra el diagrama de las interrelaciones entre estas clases.

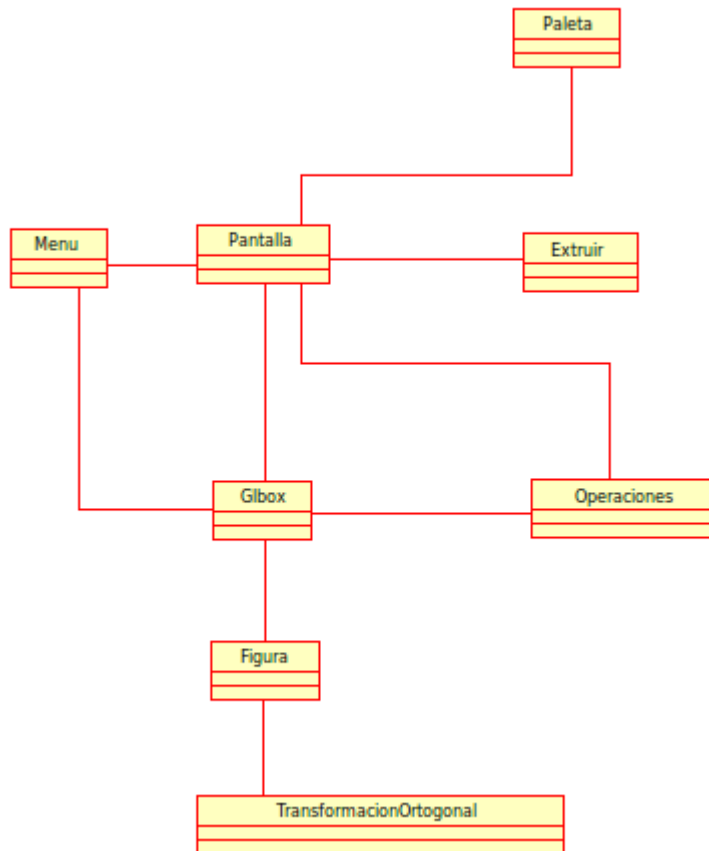


Figura 1. Interrelaciones entre las clases de la interfaz gráfica con el usuario

Visualización dinámica

Nuestra interface gráfica es dinámica ya que cuenta con diferentes herramientas que permiten manipular la vista desde diferentes posiciones con el fin de ver las geometrías simples desde diferentes perspectivas y también permite dibujar en diferentes planos de construcción XY, YZ, XZ o incluso en planos arbitrarios los cuales son combinaciones lineales de estos planos.

Las opciones para la visualización dinámica son las siguientes:

2. **Zoom.** Aumenta o reduce la vista de las geometrías que se están construyendo.
3. **Elegir Plano.** El sistema permite que se seleccione colocar la vista en alguno de los planos XY, XZ y YZ.
4. **Movimiento dinámico.** Cuando esta opción está habilitada se pueden realizar movimientos de la vista utilizando el ratón.

Construcción de una geometría simple

El ambiente cuenta con tres objetos simples, descritos a continuación, con los cuales el usuario puede construir geometrías más complejas. Una vez que el usuario activa, con ayuda del ratón, uno de estos objetos, entonces, podrá dibujarlos en la ventana principal (clase Gibox) y en el plano previamente activado por el mismo. Los tres objetos son los siguientes:

7. **Cuadrilátero.** Construye un cuadrilátero que se ajusta al tamaño que el usuario asigne a partir

del momento de hacer el primer click con el ratón y la posición final corresponde al segundo click del ratón.

8. **Circulo.** Construye una circunferencia con centro en el punto en el que el usuario hace el primer click con el ratón y con el segundo click del ratón se determina el radio.
9. **Trazado de Líneas.** Construye una colección de líneas consecutivas, con ayuda del ratón, la posición inicial corresponde al primer click y la final con el segundo click. Con una geometría construida se puede generar un plano con las líneas previamente construidas.

Extruir

Una forma de generar geometrías en tres dimensiones en esta interface gráfica es mediante el comando **Extruir**. Con este comando se genera una geometría tridimensional a partir de una geometría planar; ya que el usuario puede extender dimensión y dirección previamente seleccionadas.

En efecto, para poder extruir una geometría planar se requiere que esta geometría esté previamente seleccionada en la ventana principal y después el usuario debe hacer click **Extruir** enseguida se despliega una ventana con las opciones que existen para realizar esta operación. Una vez que el usuario selecciono la geometría, la ventana principal envía un mensaje a la clase **Gibox** para indicarle las geometrías seleccionadas y la clase **Figura** realiza la operación de extruir; después esta clase se interrelaciona con la clase **Gibox** para mostrar en la ventana principal la geometría extruida. Todo esto se muestra en la siguiente Figura 2.

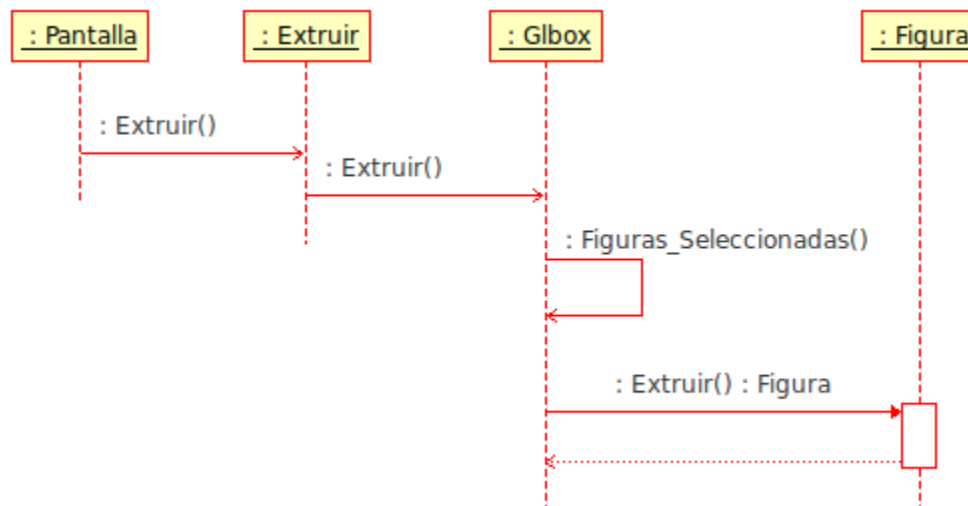


Figura 2. Proceso de extruir una geometría

Seleccionar color de la geometría

El usuario cuenta con la opción de seleccionar el color con el cual construirá las geometrías. La clase **Paleta** proporciona una variedad de colores que pueden ser seleccionados por el usuario. Una vez que en la ventana principal se activa seleccionar color se despliega la paleta de colores, y cuando el usuario ha seleccionado y aceptado un color, la clase **Gibox** establece el color que se utilizara en las siguientes geometrías que el usuario construirá (Figura 3).

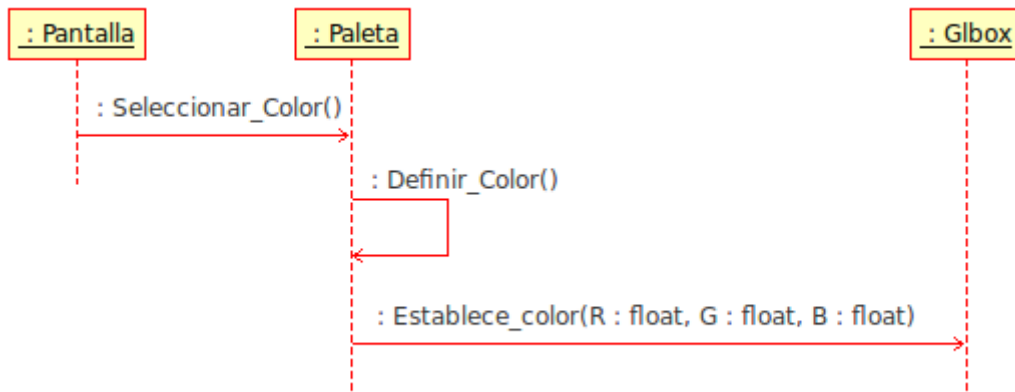


Figura 3. Proceso de selección de color.

Transformación Ortogonal

La interface gráfica cuenta con dos clases fundamentales, para más detalles ver Capítulo 2, que permiten realizar las transformaciones ortogonales ya sea con Álgebra Vectorial o Álgebra Geométrica. El funcionamiento de estas dos clases es excluyente por lo que el usuario solamente puede elegir en el menú de herramientas una de estas dos opciones. Una vez que el usuario ha elegido una opción, la clase **Gibox** almacena esta información y la construcción de la geometría se realiza mediante clase **Figura** con la opción elegida para realizar la transformación ortogonal.

Cuando el usuario aplique alguna de rotación o reflexión simple, la clase **Pantalla** envía la señal a la clase **Gibox** para que en la ventana principal el usuario construye ya sea el plano de reflexión-normal al plano o eje-bivector de rotación, entonces, la clase **Figura** es la que permite al usuario construir la geometría mediante una reflexión o rotación simple. La Figura 4 y 5 muestran el proceso de aplicar una rotación o reflexión simple con ambas álgebras.

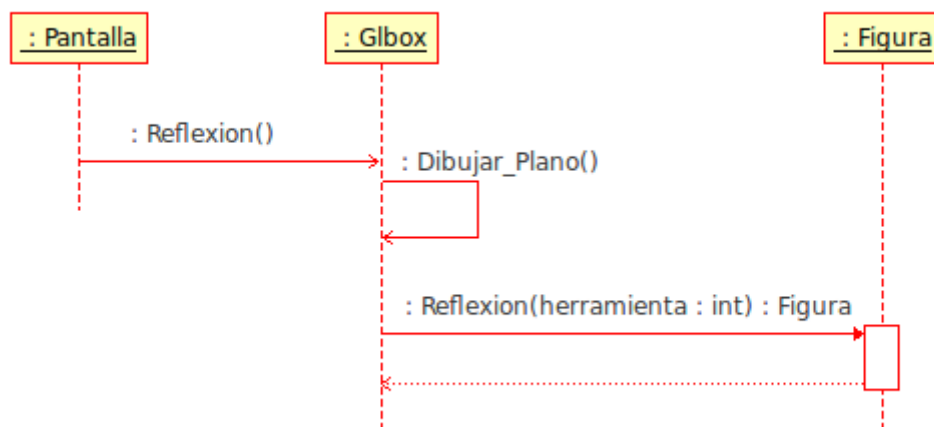


Figura 4. Proceso de aplicar una reflexión simple.

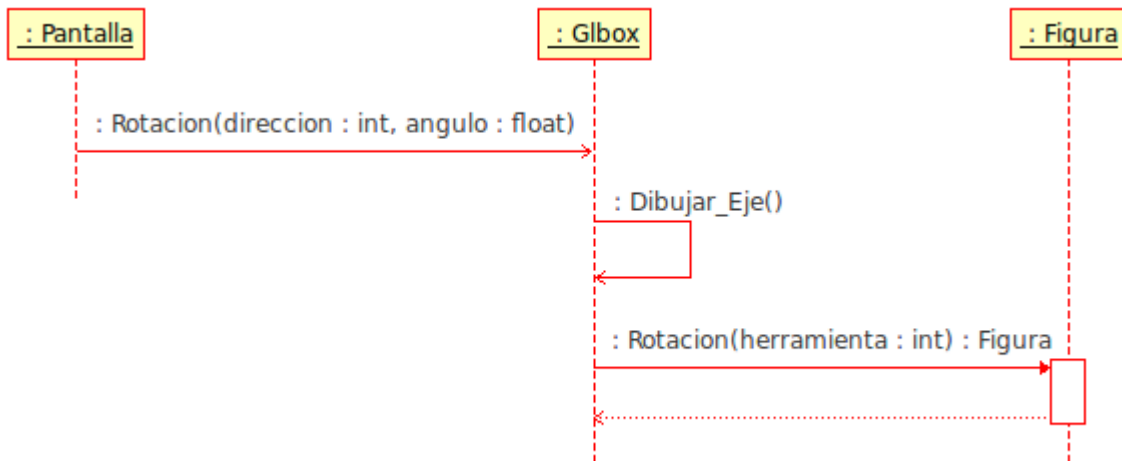


Figura 5. Proceso de aplicar una rotación simple.

Eficiencia numérica.

La clase **Operaciones** tiene como finalidad contar el número de operaciones cuando una transformación ortogonal es aplicada; un objeto de esta clase es creado en la clase **Figura** como contador de las operaciones empleadas. Después de finalizar la transformación, la clase **Operaciones** envía los resultados a la clase **Pantalla** para mostrar los resultados al usuario (Figura 6).

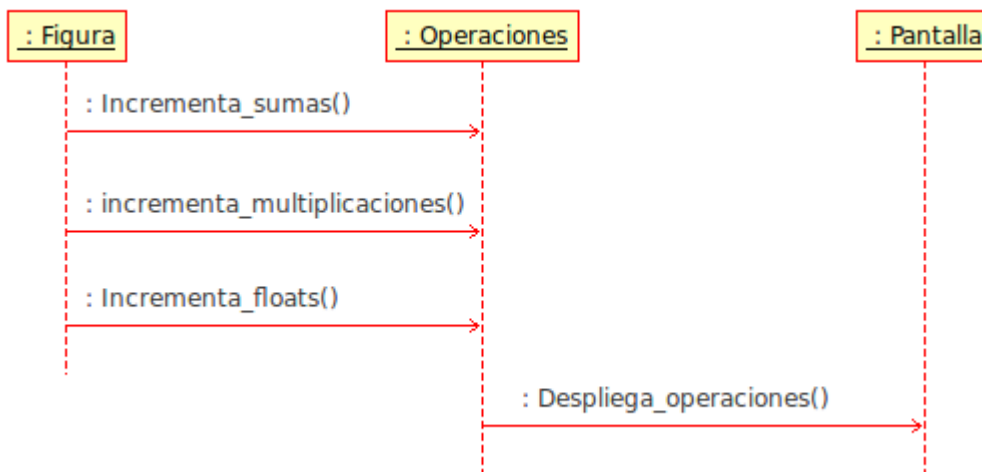


Figura 6. Proceso para contar el número de operaciones y memoria utilizada.

Ejemplo de la construcción de una geometría simple.

Con base en el manual de usuario (ver Apéndice 2), a continuación construiremos una geometría simple la cual corresponde a una antena (Figura 7). En su construcción empleamos rotaciones y reflexiones simples utilizando tanto Álgebra Vectorial como Álgebra Geométrica, con el fin de establecer una comparación de la eficiencia numérica cuando se aplica una u otra álgebra.

Para comenzar presentamos la geometría original, la cual hemos elegido como modelo para construirla con la interfaz gráfica desarrollada en este trabajo. Las Figura 7 y 8 muestra la forma y las partes que constituyen el modelo.

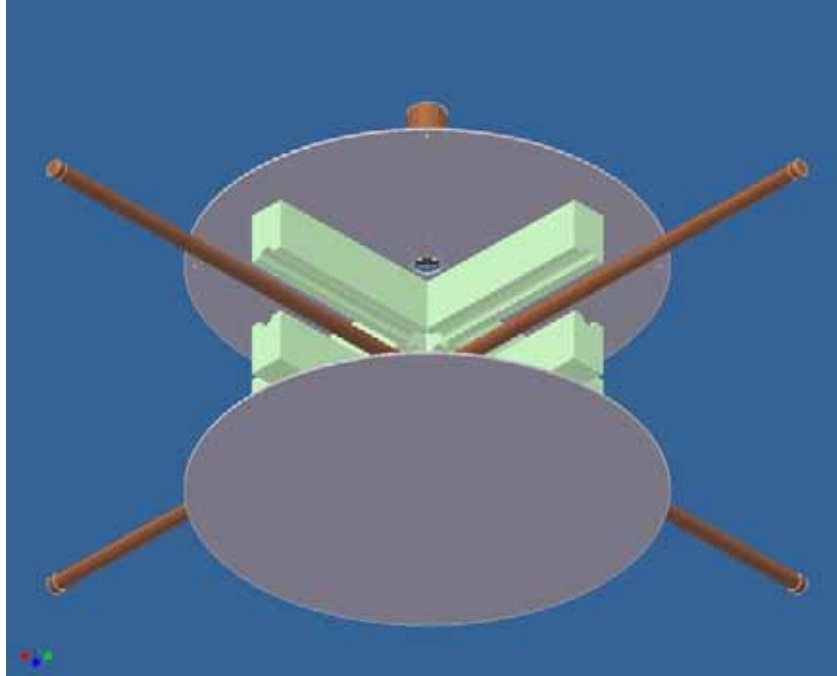


Figura 7. Una antena como modelo.

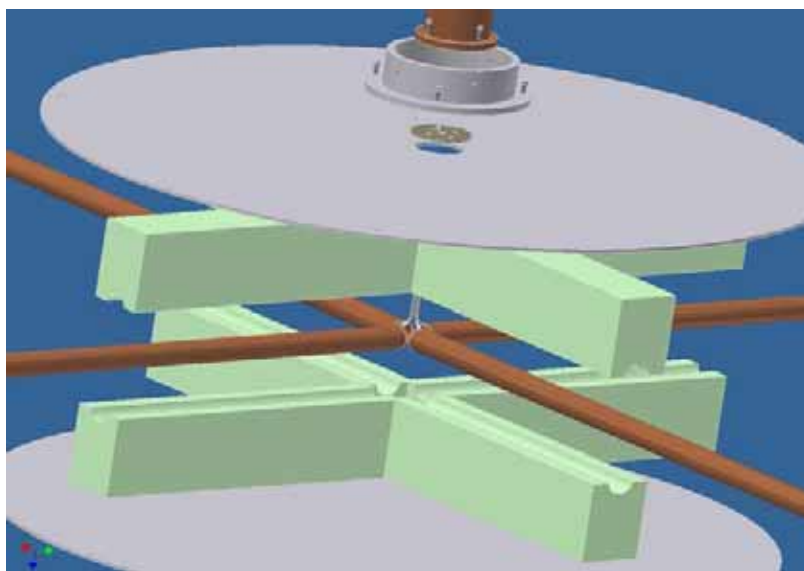


Figura 8. Otra perspectiva de la antena.

En lo que sigue se describe paso por paso su construcción de esta geometría simple.

1. Elegido previamente el color y al considerar el plano xy como plano de construcción, primero construimos el disco el cual puede ser superior o inferior mediante el botón de construir círculos el cual se encuentra en la barra de herramientas, ver Apéndice 2. Después se extruye este disco, en este ejemplo se considera una dimensión de una unidad, la dirección de la extrusión es hacia al fondo con la dirección z (ver Figura 9).

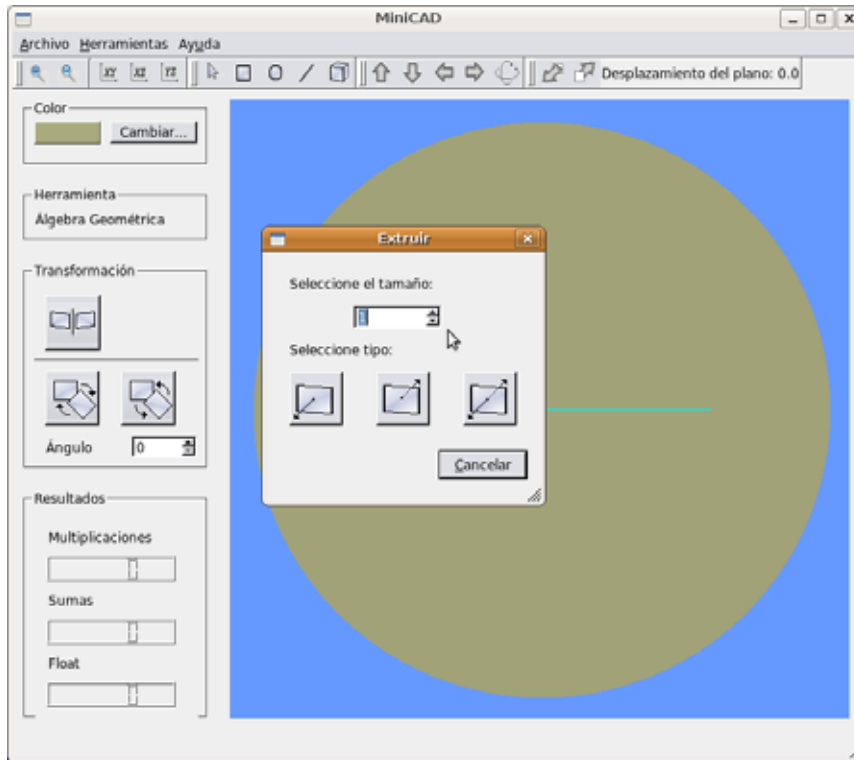


Figura 9. Construcción del disco y su extrusión en una unidad con dirección hacia el fondo.

2. Ahora se construye las barras rectangulares con otro color, el cual se obtiene de la paleta de colores. Se elige después color verde seco y con el botón de construir rectángulos, trazamos un rectángulo desde la parte superior hasta la inferior del disco y después lo extruimos con dirección hacia el frente con una dimensión de cuatro unidades (ver Figura 10). Es conveniente hacer notar que en la construcción, por ejemplo de la barra, conviene realizarla en el plano de construcción más conveniente; en este caso es un plano paralelo al disco.

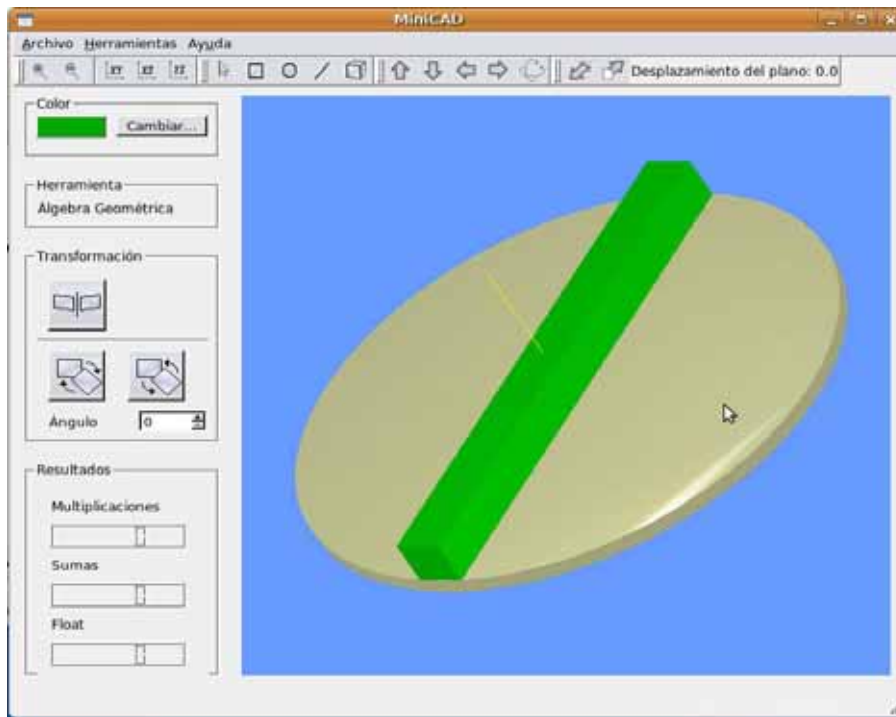


Figura 10. Construcción de la primera barra.

5. Para construir la segunda barra es suficiente aplicar una rotación de la primera barra un ángulo de 90° con eje de rotación perpendicular al plano de construcción, se elige la opción de transformación en la columna izquierda y se elige la rotación. Esto queda ilustrado en la Figura 11.

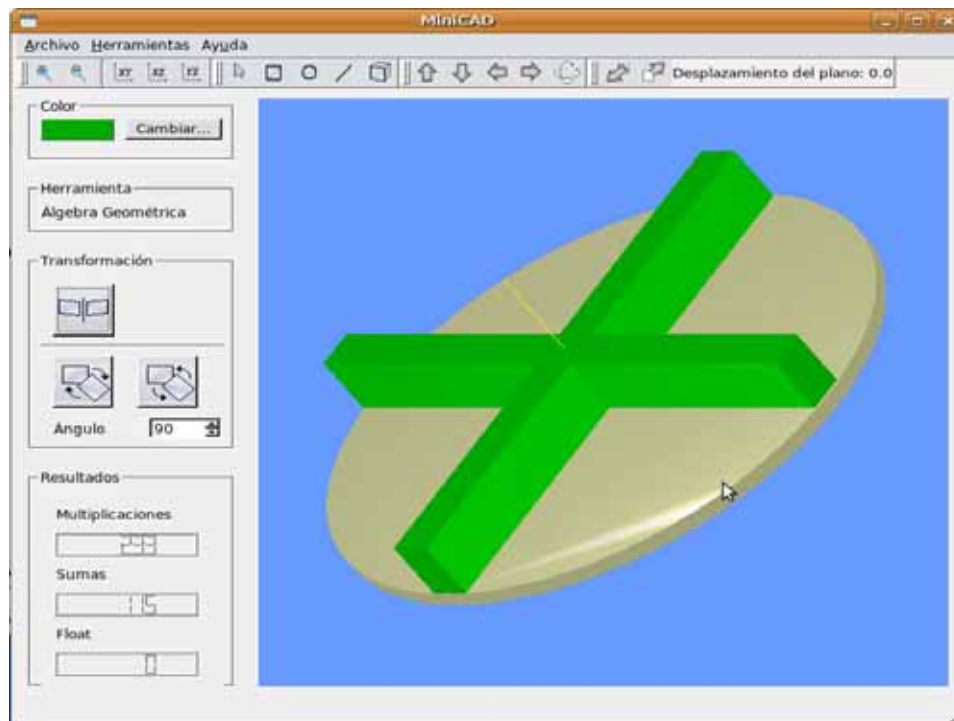


Figura 11. Construcción de la segunda barra mediante una rotación simple de 90° .

4. Ahora se construye las pequeñas barras que se encuentran en el centro de la antena (ver Figura 7 y 8) a las cuales se eligió que sean de color café. Nuevamente el plano de

construcción es el plano xy y trasladamos este plano en 0.3 con dirección hacia el frente, entonces, construimos una de las pequeñas con color café (ver Figura 12).

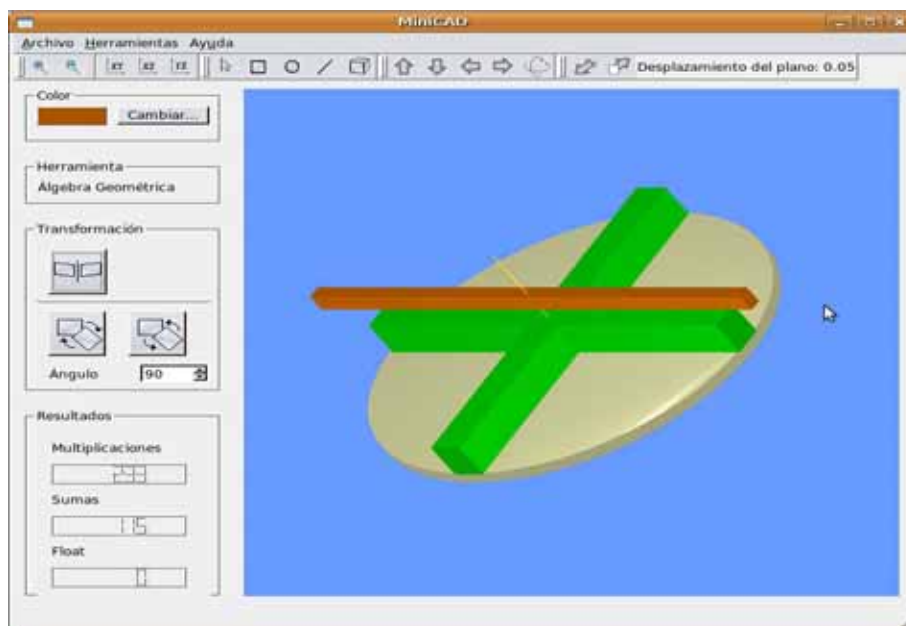


Figura 12. Construcción de la pequeña barra intermedia de la antena.

6. Análogamente, para la construcción de la segunda pequeña barra de la Figura 4, realizamos otra rotación de 90° alrededor del eje perpendicular al plano de construcción (ver Figura 13).

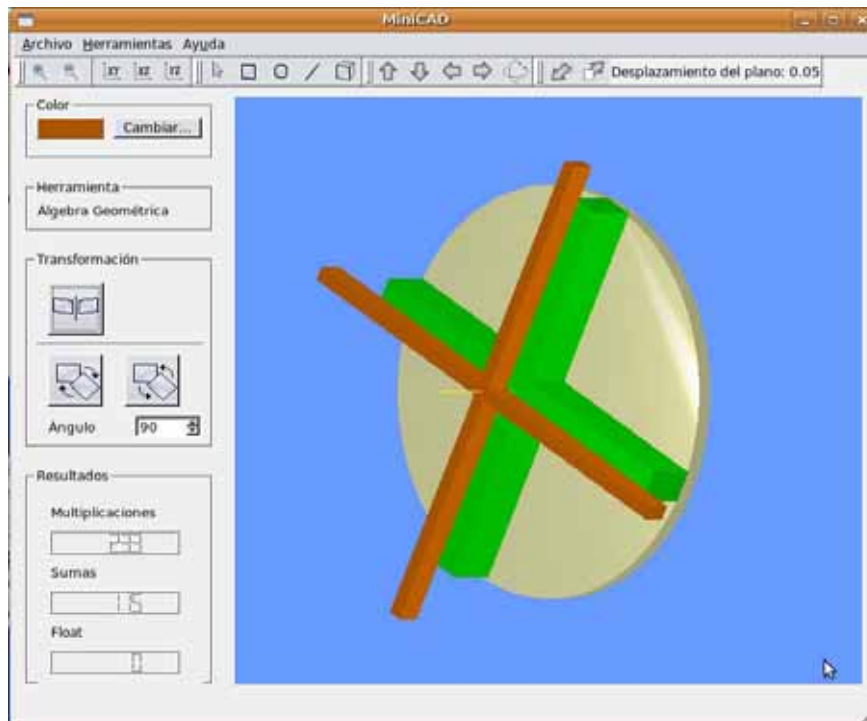


Figura 13. Construcción de la segunda pequeña barra intermedia de la antena mediante una rotación de 90° .

6. Ahora para obtener el segundo disco que contenga sus respectivas barras de color verde,

seleccionamos el disco junto con las dos barras, mediante el botón de selección. Después se elige la opción de transformación de la columna izquierda y seleccionamos reflexión. Enseguida construimos el eje de reflexión justo en la mitad de las barras pequeñas internas de color café. Para visualizar este eje cambiamos la vista al seleccionar como plano de construcción al plano **yz**, mediante el botón correspondiente. El resultado obtenido es la reflexión especular respecto a un plano con normal -el eje construido- después de haber elegido la opción de transformación en la columna izquierda para dicha reflexión. Las Figuras 14 y 15 ilustran la construcción del segundo disco.

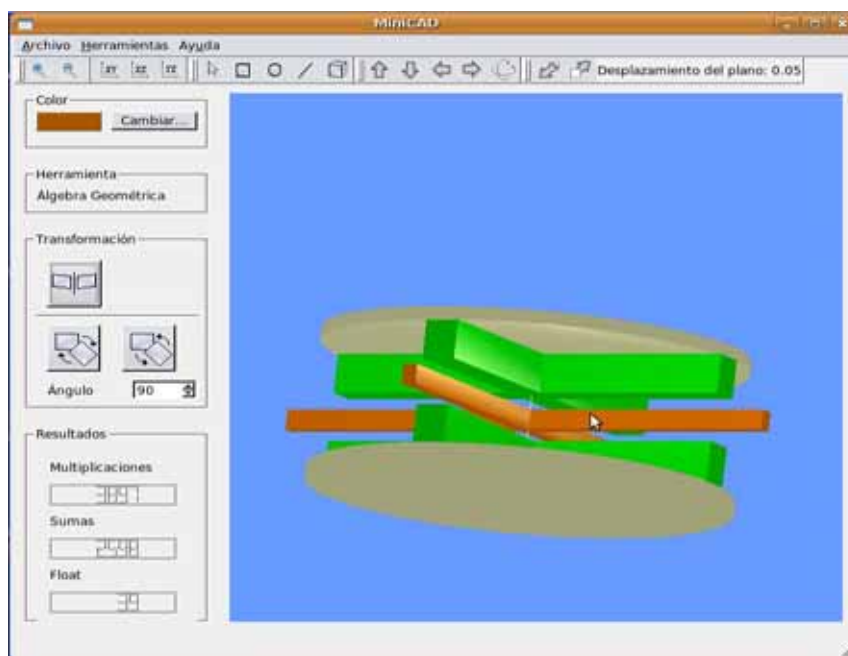


Figura 14. Construcción del segundo disco con sus barras con base en una reflexión.

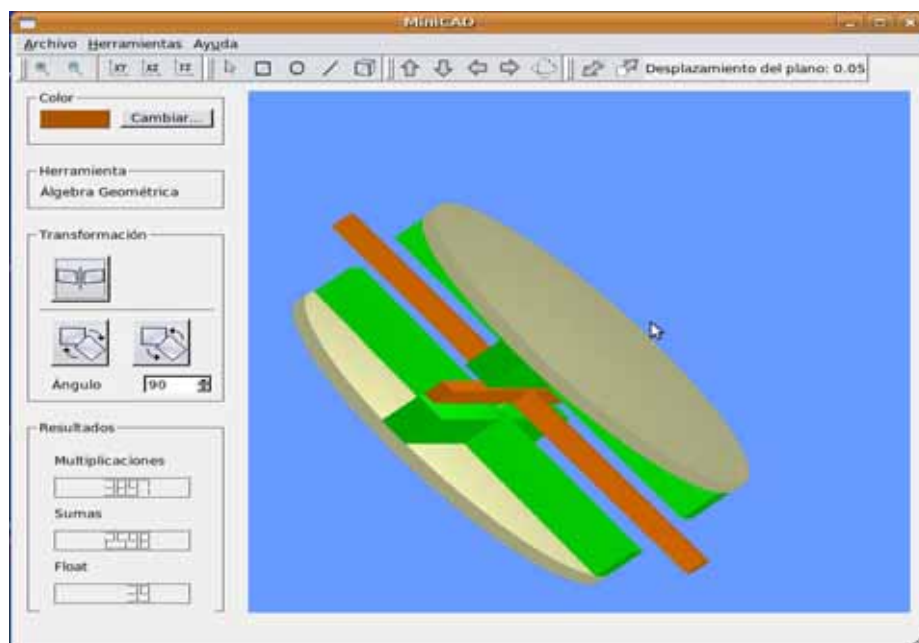


Figura 15. Otra perspectiva de la construcción del segundo disco con sus barras mediante una reflexión.

7. Finalmente, se construye el cilindro que se encuentra en la parte superior de la antena (ver Figura 7). Para realizarlo se considera nuevamente el plano xy como plano de construcción, se usa como referencia el centro del disco superior. Se construye un círculo y lo extruimos con dirección hacia el frente en seis unidades. La Figura 15 muestra la construcción de la geometría completa.

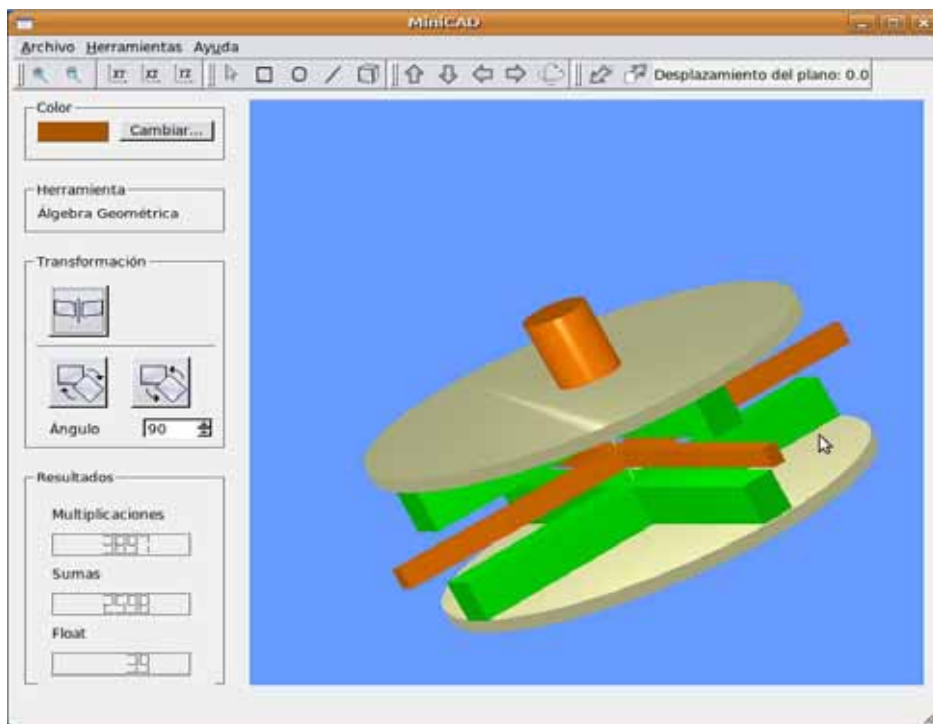


Figura 16. Construcción completa de la antena mediante nuestro ambiente visual de programación.

Claramente la construcción anterior se realizó con ambas álgebras para así poder establecer su comparación numérica cuando se construye con una u otra álgebra.

Comparación numérica

En la comparación solamente hemos considerado para construir la antena de la Figura 7, el número de multiplicaciones y sumas realizadas en su construcción con ambas álgebras. A continuación presentamos una tabla comparativa de esta construcción.

| Transformación Ortogonal | Álgebra Vectorial | | Álgebra Geométrica | |
|--------------------------------|-------------------|------------------|--------------------|------------------|
| | Sumas | Multiplicaciones | Sumas | Multiplicaciones |
| 1er. Rotación (Figura 11) | 84 | 135 | 115 | 233 |
| 2da. Rotación (Figura 13) | 84 | 135 | 115 | 233 |
| Reflexión (Figuras 14 y 15) | 3681 | 3735 | 2598 | 3897 |

Tabla 1. Comparación numérica entre el álgebra lineal y el álgebra vectorial, al construir la antena.

Conclusiones

El ambiente de programación visual desarrollado en este proyecto funciona como un sistema CAD con la ventaja de incluir además AG. El AG provee un lenguaje unificado para representar operaciones y transformaciones geométricas. Específicamente, las transformaciones ortogonales resultan de combinar vectores de acuerdo a su producto geométrico, es decir, son elementos del álgebra, como fue desarrollado en el Capítulo 1. Con esta representación elegante y compacta de las reflexiones especulares y como toda transformación ortogonal es la composición de un número finito de reflexiones especulares (Teorema de Cartan del Apéndice 1), entonces, cualquier transformación ortogonal es resultado de un producto geométrico finito entre vectores y sus inversos.

En contraste, el AV para realizar las transformaciones presentó problemas en el software en la distinción entre vectores y puntos; ya que los vectores son direcciones. Esto fue resuelto al introducir un origen respecto al cual los vectores de posición representan los puntos. Esta diferencia semántica entre un vector como dirección, un vector como punto y un vector axial fueron considerados y representados, en el capítulo 2, mediante las clases **vector** y **Point3D**. En estas clases se tuvo que considerar que los vectores, como puntos, se transforman en forma diferente a los vectores axiales o las direcciones. Otro problema que se tuvo que resolver fue cuando se consideran otros objetos geométricos como son los planos, los cuales con AV son obtenidos con el vector axial –normal al plano. En cambio con AG, obtener este vector axial es simplemente multiplicar el bivector –plano- por el pseudoescalar del espacio 3D.

Por otra parte, aplicar matrices ortogonales a puntos, vectores y planos exigió diferentes implementaciones que tuvieron que programarse para cada caso por separado. Aún más, emplear matrices para manipular rotaciones no es tan conveniente para codificar las propiedades geométricas esenciales tales como el eje y el ángulo. Con AG son solamente operaciones algebraicas y su codificación resultó más sencilla de implementar. En los diagramas UML de las Figuras 2 (AV) y 5 (AG) y la programación de la clase **TransformaciónOrtogonal**, para ambas Álgebras analizadas en el capítulo 2 pueden distinguirse ventajas del AG en el desarrollo de nuestro sistema CAD.

En el capítulo 3 mediante diagramas UML se describieron los procesos internos de aplicar una transformación ortogonal con ambas Álgebras (Figuras 4 y 5) y de la interfaz gráfica con el usuario (Figura 1). En la construcción de una geometría simple (antena) mediante dos rotaciones simples (Figuras 11 y 13) y una reflexión simple (Figuras 14 y 15), encontramos que una de las ventajas numéricas del AG fue en el número de sumas realizadas en la reflexión simple (ver Tabla 1). En los otros resultados numéricos obtenidos, el AV presentó ventajas numéricas.

Aunque, lo anterior no es del todo determinante puesto que se tendrá que realizar una experimentación numérica más amplia y aplicar otros criterios de eficiencia numérica para poder obtener resultados numéricos más finos para poder decidir si el AG es más conveniente que el AV. Este trabajo es para desarrollar en el futuro. Sin embargo, los objetivos establecidos para este proyecto fueron cubiertos satisfactoriamente.

Finalmente, el resultado fundamental del proyecto desarrollado es proveer de un sistema CAD el cual puede ser aplicado y extendido por las nuevas generaciones de ingenieros interesados en el uso de nuevas herramientas y la optimización numérica de sistemas CAD/CAM.

Bibliografía

- [1] S. Grossman. (2008), *Álgebra Lineal*. Grupo Editorial Iberoamerica, 2da. Edición.
- [2] L. Dorst, D. Fontijne y S. Mann. (2007) *Geometric Algebra for Computer Sciencie*. Elsevier.
- [3] G. Aragón-González, J.L. Aragón, M.A. Rodríguez-Andrade and L. Verde-Star(2009), *Reflections, Rotations, and Pythagorean Numbers*. Adv. appl. Clifford alg. **19** 1–14.
- [4] R. Goldman. (2003), *Deriving Linear Transformations in Three Dimensions*. IEEE Comp. Graph. And Appl. 66-71.
- [5] L. Joyanes. (2007), ***Estructura de Datos en C++***. Mcgraw-Hill, 1er. Edición.
- [6] S. Schach. (2005), *Análisis Y Diseño Orientado A Objetos Con Uml Y El Proceso Unificado*. Mcgraw-Hill.
- [7] G. Golub and C. Van Loan. (1996), *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences, 3rd Edition.
- [8] D. Lilja. (2000), *Measuring computer performance*. Department of Electrical and Computer Engineering University of Minnesota.
- [9] D. Hearn y P. Baker. (2007), *Gráficos Por Computadora Con Opengl*. Pearson Educación, 3er. Edición.
- [10] J. Blanchette y M. Summerfield. (2006), *C++ GUI Programming with Qt 3*. Prentice Hall.
- [11] J. Su-Chen. (1994), *Computer Numerical Control: From programming to Networking*. Delmar Publishers.

Apéndice 1

Aquí presentamos la demostración del teorema de Cartan para 3D la cual está basada en la demostración dada en [3] para n-dimensional.

En efecto, sea \mathbf{T} una transformación ortogonal, $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ una base ON de 3D y consideremos a \mathbf{e}_1 y $\mathbf{T}(\mathbf{e}_1)$. Si $\mathbf{T}(\mathbf{e}_1) = \mathbf{e}_1$, entonces, la reflexión simple es la identidad. Si no se cumple entonces la reflexión simple es:

$$-\mathbf{n}_1 \mathbf{v} \mathbf{n}_1^{-1}$$

con $\mathbf{n}_1 = \mathbf{T}(\mathbf{e}_1) - \mathbf{e}_1$. Claramente se satisface que:

$$-\mathbf{n}_1 \mathbf{T}(\mathbf{e}_1) \mathbf{n}_1^{-1} = \mathbf{e}_1$$

y $-\mathbf{n}_1 \mathbf{T}(\mathbf{e}_i) \mathbf{n}_1^{-1}; i = 2, 3$ pertenece al plano cuya normal es \mathbf{e}_1 . Consideremos ahora a $-\mathbf{n}_1 \mathbf{T}(\mathbf{e}_2) \mathbf{n}_1^{-1}$ y \mathbf{e}_2 . Si $-\mathbf{n}_1 \mathbf{T}(\mathbf{e}_2) \mathbf{n}_1^{-1} = \mathbf{e}_2$ la reflexión simple es la identidad. Si no se cumple entonces la reflexión simple es:

$$\mathbf{n}_2 \mathbf{n}_1 \mathbf{v} \mathbf{n}_1^{-1} \mathbf{n}_2^{-1}$$

con $\mathbf{n}_2 = -\mathbf{n}_1 \mathbf{T}(\mathbf{e}_2) \mathbf{n}_1^{-1} - \mathbf{e}_2$. Claramente se satisface:

$$\mathbf{n}_2 \mathbf{n}_1 \mathbf{T}(\mathbf{e}_2) \mathbf{n}_1^{-1} \mathbf{n}_2^{-1} = \mathbf{e}_2$$

y $\mathbf{n}_2 \mathbf{n}_1 \mathbf{T}(\mathbf{e}_3) \mathbf{n}_1^{-1} \mathbf{n}_2^{-1}$ pertenece a la línea generada por \mathbf{e}_3 . Finalmente consideramos $\mathbf{n}_2 \mathbf{n}_1 \mathbf{T}(\mathbf{e}_3) \mathbf{n}_1^{-1} \mathbf{n}_2^{-1}$ y \mathbf{e}_3 . Y si son diferentes entonces la reflexión simple es:

$$-\mathbf{n}_3 \mathbf{n}_2 \mathbf{n}_1 \mathbf{v} \mathbf{n}_1^{-1} \mathbf{n}_2^{-1} \mathbf{n}_3^{-1}$$

y el resultado final sería:

$$\mathbf{T}(\mathbf{v}) = -\mathbf{n}_3 \mathbf{n}_2 \mathbf{n}_1 \mathbf{v} \mathbf{n}_1^{-1} \mathbf{n}_2^{-1} \mathbf{n}_3^{-1}$$

Apéndice 2

Manual del usuario

Instalación y ejecución del programa.

El presente manual es una guía para que el usuario pueda interactuar con el ambiente de visualización gráfica (interfaz gráfica, ver Glosario) en la construcción de geometrías simples (figuras geométricas construidas con base en puntos, líneas, rectángulos y círculos). En la construcción puede usarse rotaciones y reflexiones simples y se podrá comparar la eficiencia numérica en cuanto a número de multiplicaciones y sumas ya sea con Álgebra Vectorial o Álgebra Geométrica. El usuario podrá dibujarlas siguientes primitivas: puntos, líneas, círculos y rectángulos y a partir de ellos formar otras geometrías combinadas con estas primitivas.

La interfaz gráfica fue implementada con el lenguaje de programación C++ ([5] y [10]) y la biblioteca multiplataforma (ver Glosario) **Qt** [10] la cual es una de las más apropiadas para desarrollar interfaces gráficas con el usuario con el lenguaje C++. Las aplicaciones basadas en **Qt** utilizan botones, ventanas, etc., que permiten la interacción con el usuario visualizando las funciones que proporciona una aplicación en particular. Estos conjuntos de herramientas son los indicados para desarrollar aplicaciones gráficas que se ejecuten sobre una interfaz X-Window (ver Glosario) en sistemas Unix. La biblioteca **Qt** también está disponible para los sistemas operativos de Microsoft Windows.

Aunque existen otras herramientas para crear interfaz gráficas, la razón de seleccionar a **Qt** y C++ es su flexibilidad y dado que su ejecución es más rápida lo cual es muy sustancial en cualquier sistema CAD (ver Glosario) debido a la gran cantidad de cálculos numéricos y velocidad indispensables.

Es importante leer este manual antes de realizar cualquier operación con la interfaz gráfica descrita en este manual. Una vez que se haya leído y comprendido este manual, es conveniente tenerlo siempre a la mano para resolver cualquier duda o emergencia que se presente.

La interface gráfica PDPA-CAD9 puede ejecutarse en la plataforma UBUNTU 8.10 en la cual debe estar instalada la biblioteca multiplataforma **Qt** para su buen funcionamiento, Si está biblioteca no está instalada a continuación indicamos los pasos a seguir para que el usuario pueda instalarla:

1. Entrar a UBUNTU y luego a Synaptic: Sistema -> Administración -> Gestor de Paquetes Synaptic.
2. Buscar y seleccionar **qt3 designer** y hacer click en Aplicar para instalar.
3. Buscar y escribir **opengl qt3**, después seleccionar: **libqwtplot3d-qt3**, y nuevamente añadir para instalar.

Finalmente,

4. Ejecutar PDPA-CAD9 y se despliega la interfaz gráfica.

Interfaz Gráfica

La Figura 1 muestra la interfaz gráfica desarrollada para realizar comparaciones numéricas al realizar reflexiones y rotaciones simples cuando se utiliza Álgebra Vectorial y Álgebra geométrica,

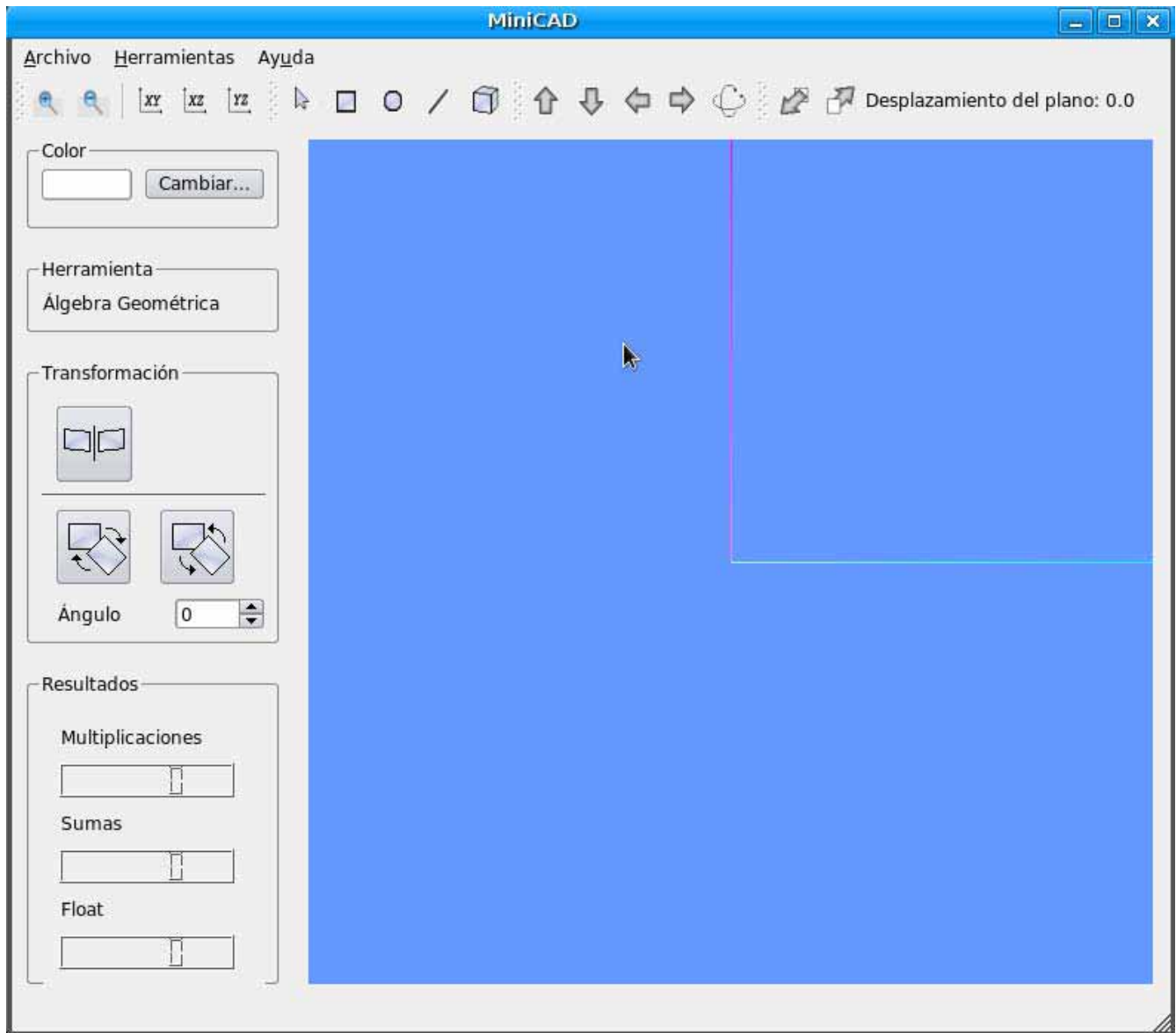


Figura 1. Interfaz Gráfica.

A continuación describimos cada uno de los componentes de la interfaz.

Componentes de la interfaz gráfica

Menús

La interfaz gráfica dispone de menús desplegables a los que se accede mediante la barra de menús situada en la parte superior de la ventana de la interfaz, se encuentran disponibles tres menús: Archivo, Herramientas y Ayuda (la Figura 2).

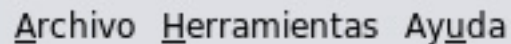
A horizontal grey bar containing three menu items: 'Archivo', 'Herramientas', and 'Ayuda'. Each item has a small underline under the first letter.

Figura 2. Barra de menús.

Para utilizar un menú.

- En la barra de menús, hacer click, con el botón izquierdo del ratón, en el nombre de un menú para visualizar la lista de opciones. En el menú, pulse una opción o bien utilice flecha ↓ para desplazarse por la lista y, a continuación, pulse Intro-enter.
- Otra opción para acceder a los menús es pulsando la tecla Alt y la letra subrayada en el nombre del menú. A continuación, pulse la letra subrayada en el nombre de la opción. Por ejemplo, para hacer un dibujo nuevo, pulse la tecla Alt y la letra A para abrir el menú Archivo. A continuación, elija la opción Nuevo.

Enseguida describimos que hacen cada una las opciones que están dentro de los menús:

1. Archivo

Hay dos opciones:

- Nuevo: Permite crear un nuevo dibujo, limpiando la pantalla de dibujo de todo lo que anteriormente se ha hecho.
- Abrir: Permite abrir una geometría construida previamente con extensión *.cad.
- Guardar: Permite almacenar las geometrías construidas en un archivo con extensión *.cad.
- Salir: Esta opción nos permite salir de la interfaz gráfica.

2. Herramientas

Hay dos opciones:

- Álgebra Vectorial: Al seleccionar esta opción podemos hacer uso de reflexiones o rotaciones con Álgebra lineal sobre una figura ya construida.
- Álgebra Geométrica: Al seleccionar esta opción podemos hacer uso de reflexiones o rotaciones con Álgebra geométrica sobre una figura ya construida.

3. Ayuda

- La ayuda es una opción que despliega información para resolver dudas o ver el funcionamiento específico de la interfaz.

Barra de herramientas

La barra de herramientas, ubicada debajo de la barra de menús, está organizada con botones que corresponden a los comandos principales como son: construir círculos construir rectángulos extruir, efectuar visualización dinámica, etc., de acuerdo al estándar utilizado para ubicar los componentes en una interfaz (Figura 3).



Figura 3. Barra de herramientas de la interfaz gráfica.

Ahora describiremos cada una de los comandos-botones de la barra de herramientas:

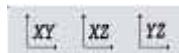
Zoom

Con el zoom podemos acercar o alejar a algún objeto ya construido dibujado en el área de trabajo. En la barra de herramientas encontramos el zoom con los dos siguientes botones.



Planos

Los planos nos ayudan a ver la geometría desde tres planos distintos: **xy**, **xz**, y **yz**. Por ejemplo, si dibujamos una geometría con plano de construcción **xy** y después visualizarla por el costado-atrás hay que seleccionar el plano **xz**. Los botones para usar estas opciones son los siguientes.



Seleccionar

Esta opción ayuda a seleccionar un parte específica o toda la geometría. Es de gran utilidad sobre todo cuando se han construidos varios objetos y solo a uno de ellos queremos extruir o aplicarle una transformación ortogonal, esto lo podemos hacer mediante al hacer click en el botón-seleccionar, con el botón izquierdo del ratón, después hacer click en el lugar área que ocupa el objeto que queremos seleccionar y finalmente aplicar lo deseado. El botón es el siguiente.



Construir geometrías simples

Las primitivas para construir geometrías que tiene la interface gráfica son: rectángulos, círculos y líneas; con las cuales y al aplicar rotaciones y reflexiones simples podemos construir geometrías más complejas. Con los siguientes botones construimos son con los que construimos: rectángulos, círculos y líneas.



Extruir

La extrusión aumenta las dimensiones de una figura en la dirección que el usuario haya indicado, Por ejemplo, construir un cuadrado en el plano de construcción **xy** (Figura 4).

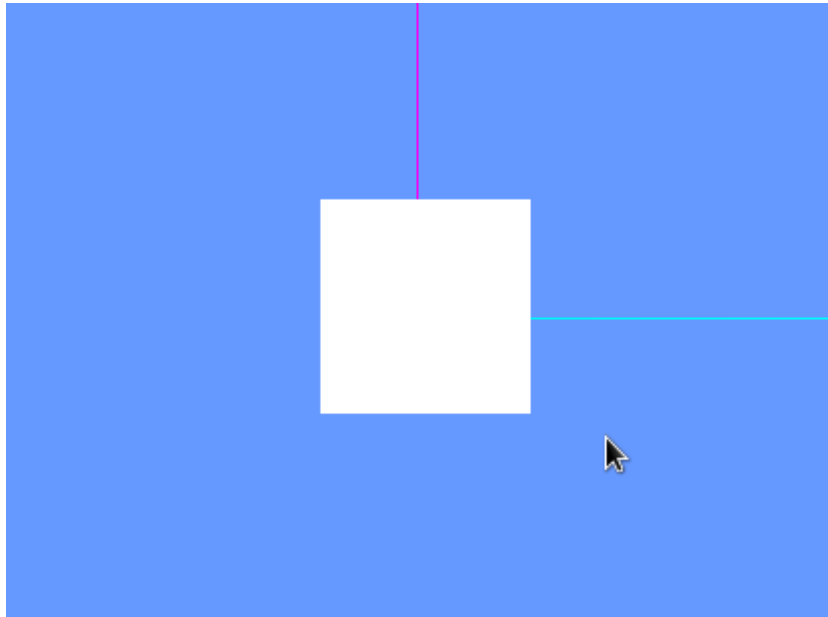


Figura 4. Cuadrado construido antes de extruir.

Después al hacer la extrusión por una unidad, el cuadrado queda como se ve en la Figura 5.

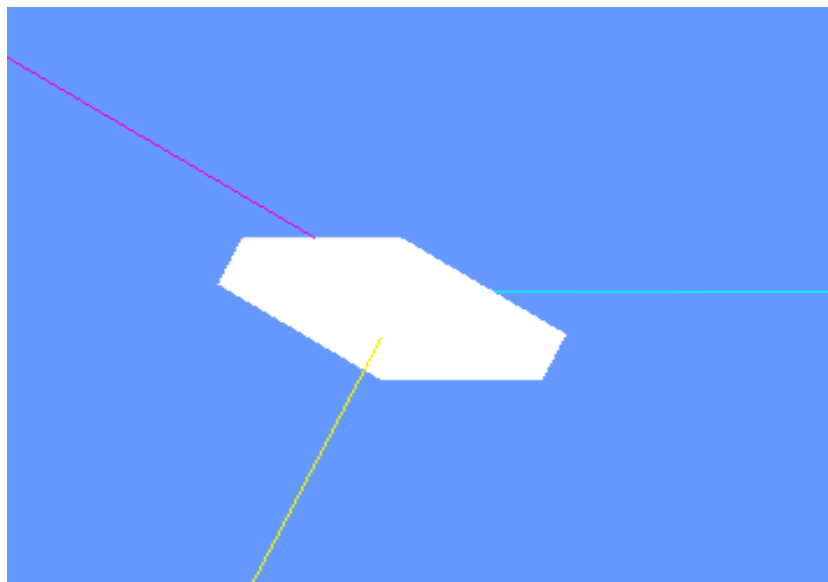


Figura 5. Cuadrado extruido una unidad.

Hay tres formas de realizar la extrusión. La primera extruye la geometría hacia el fondo como es el caso de la Figura 5, la segunda extruye la geometría hacia adelante y la tercera extruye la geometría hacia el fondo y adelante. Para elegir alguna de estas tres opciones se despliega una ventana (Figura 6) que aparece después de hacer click al botón de extrusión:

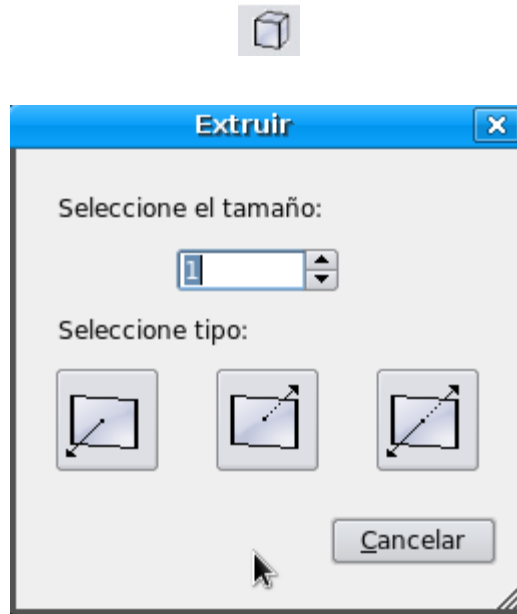


Figura 6. Ventana para elegir el modo de extrusión en una geometría.

Visualización dinámica.

Visualización dinámica proporciona las diferentes perspectivas con las que podemos ver la geometría construida. Podemos rotar la geometría y ver, por ejemplo, la parte superior, inferior o el frente, etc. La visualización dinámica puede hacerse de formas:

La primera con los botones que tienen flechas:



Al hacer click en uno de estos botones podemos mover la geometría hacia arriba, abajo izquierda y derecha.

La segunda forma es mediante el ratón con el botón:



Al hacer click en este botón podemos mover la geometría simplemente con mantener presionado el botón izquierdo del ratón y entonces movemos el ratón en la dirección que queremos mover la geometría.

Desplazamiento del plano de construcción.

Esta es una opción útil ya que traslada el plano de construcción hacia adelante o hacia atrás el número de unidades especificadas por el usuario. Tiene la finalidad de no tener geometrías traslapadas o construir geometrías que necesitan ser construidas en el mismo plano de construcción, v. gr. **xy**. Por ejemplo, si el usuario quiere mantener la altura del plano de construcción la traslación se realiza en el eje **y**.

En la Figura 7 se ha trasladado el plano de construcción con el fin de no traslapar las geometrías.

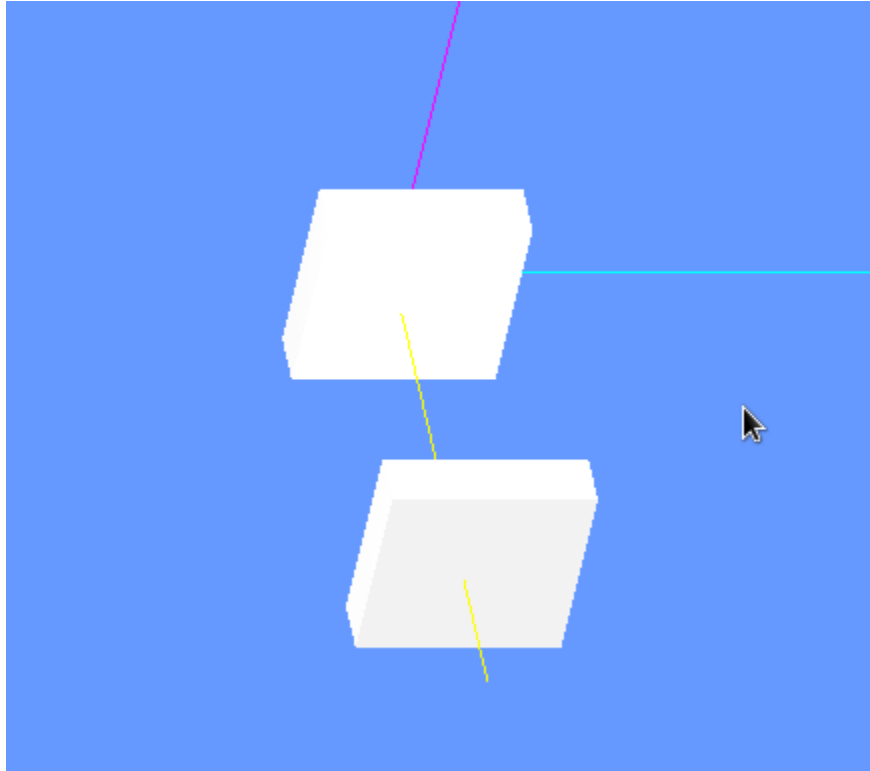
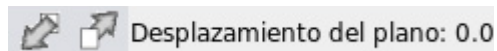


Figura 7. Traslación del plano de construcción.

Los botones que usamos para la traslación hacia adelante o hacia el fondo son los siguientes



Los botones tienen como iconos flechas que apuntan adelante y atrás respectivamente y al lado derecho de estos botones está una etiqueta que dice Desplazamiento del plano y hay un valor numérico el cual indica el desplazamiento que se está realizando cuando se hace click en las flechas.

A continuación presentamos las opciones de la parte izquierda del área de trabajo de la interfaz gráfica.

Color

Este comando permite al usuario cambiar el color con el cual las geometrías serán construidas.

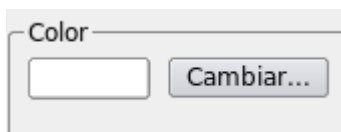


Figura 8. Color.

Con este botón el usuario puede asignar el color a la geometría. Para cambiar el color por defecto el usuario tiene que hacer click en el botón Cambiar el cual se encuentra a la izquierda (Figura 8) para desplegar la paleta de colores (Figura 9) en donde el usuario podrá seleccionar el color; después se hace click en el botón aceptar o cancelar en ese caso se queda establecido el color anterior.



Figura 9. Paleta de colores.

Cuando el usuario ha seleccionado un color: el recuadro originalmente de a color blanco de la Figura 8 cambia al color que haya elegido (Figura 10).

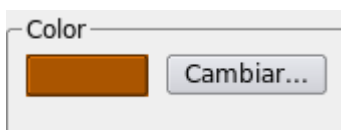


Figura 10. Selección de color.

Herramienta

Este enunciado muestra al usuario la herramienta algebraica en uso cuando aplique alguna transformación ortogonal: álgebra Vectorial o álgebra geométrica. El enunciado cambia cuando el usuario cambia la opción en el menú Herramientas de la barra de menús (Figura 11).

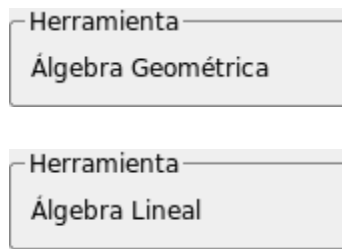


Figura 11. Selección de herramienta algebraica.

Transformación

En este lugar el usuario puede elegir la transformación ortogonal que quiere aplicar a la geometría construida (Figura 12).



Figura 12. Selección de la transformación ortogonal.

La reflexión simple

El usuario debe tener seleccionada la parte o toda la geometría construida antes de aplicar una reflexión simple. Para aplicarla el usuario tiene que hacer click en el botón superior de la Figura 12. Después de hacer la selección el usuario tiene que trazar en el área de trabajo el eje (una línea) a través del cual se hará la reflexión (Figura 13).

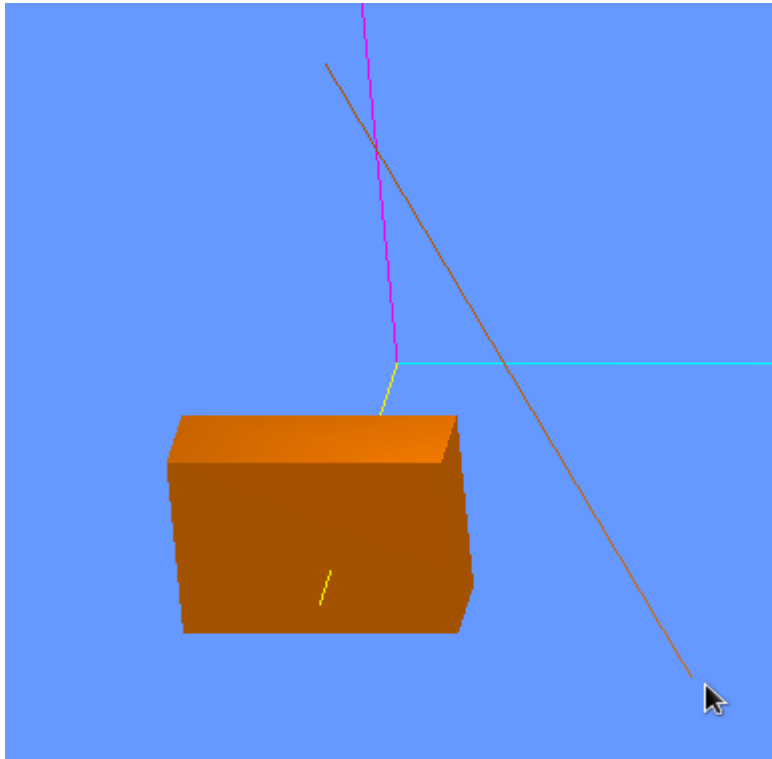


Figura 13. Trazado del eje de reflexión.

La Figura 14 muestra la geometría reflejada a través del eje anteriormente trazado.

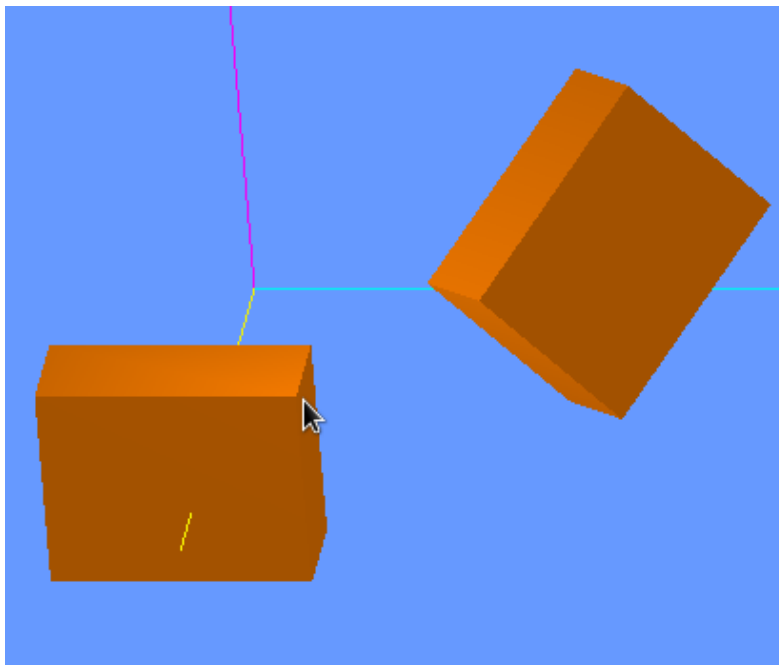


Figura 14. Reflexión de una geometría simple.

La rotación simple

El usuario debe tener seleccionada la parte o toda la geometría construida antes de aplicar una rotación simple. Para la rotación simple se emplean los dos botones inferiores de la Figura 12. El de

la izquierda rota la geometría construida un ángulo, especificado por el usuario, alrededor de un eje, previamente trazado, en sentido negativo (según las manecillas del reloj) y el botón de la derecha rota la geometría construida pero en sentido positivo (contrario a las manecillas del reloj), El usuario tiene que introducir el ángulo, en grados, en el recuadro que dice Ángulo. Análogamente, como en la reflexión, cuando el usuario hace clic en cualquiera de estos los dos botones tiene que trazar el eje de rotación en el área de trabajo (ver Figura 15).

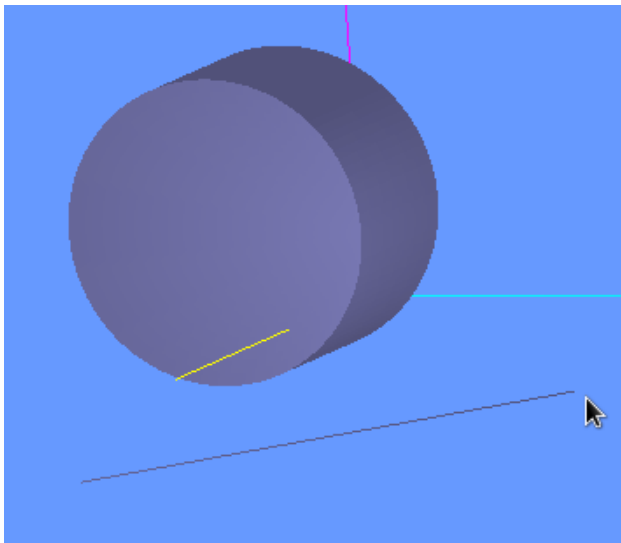


Figura 15. Trazado del eje de rotación.

Como ejemplo podemos rotar la Figura 15 un ángulo de 90° para obtener la Figura 16.

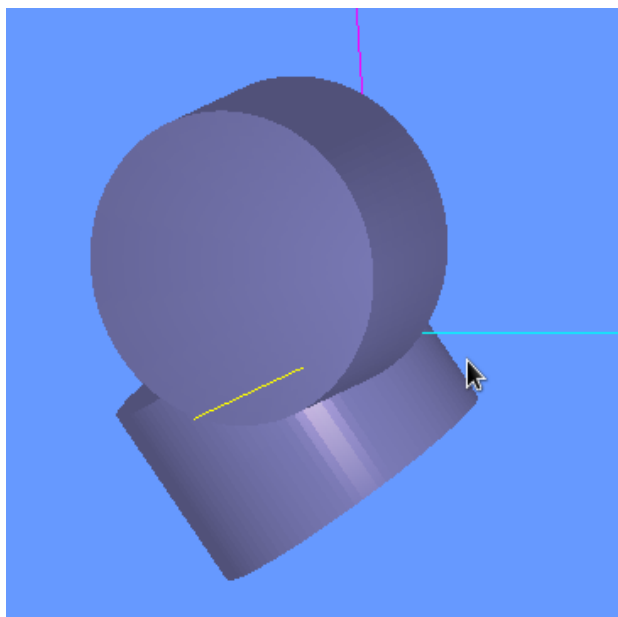


Figura 16. Rotación de una geometría alrededor del eje dado por un ángulo de 90° .

Resultados

En el recuadro de Resultados el usuario puede comparar numéricamente el número de multiplicaciones, sumas y variables flotantes utilizadas cuando en la construcción de la geometría ha utilizado transformaciones ortogonales y poder así analizar cuando es más conveniente realizar estas transformaciones con Álgebra Vectorial o Álgebra Geométrica. Por ejemplo, la Figura 16 muestra los resultados de hacer la rotación con Álgebra Geométrica de la geometría de la Figura 15.

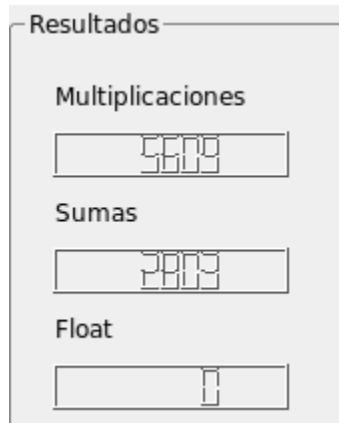


Figura 16. Resultado de operaciones al rotar la geometría de la Figura 15.

Borrar una geometría

Para borrar una geometría simplemente el usuario tiene que seleccionar la parte o toda la geometría y oprimir el botón Supr en el teclado.

Glosario

Interfaz gráfica: Para simplificar al usuario el uso de computadoras es habitual utilizar metáforas visuales como es la denominada interfaz gráfica de usuario (IGU ó GUI, por sus siglas en inglés). Por lo que con una interfaz gráfica, el usuario puede fácilmente interactuar y establecer contacto más intuitivo con una computadora.

Multiplataforma: Programa o dispositivo que puede utilizarse en distintas plataformas de hardware y/o distintos sistemas operativos.

X-Window: El sistema de ventanas X (X Window System, en inglés) fue desarrollado a mediados de 1980 en el MIT (Instituto Tecnológico de Massachusetts, por sus siglas en inglés) con la finalidad de dotar de una interfaz gráfica al usuario.

CAD: Computer Assisted Draw, Diseño Asistido por Computadora (por sus siglas en inglés)

Apéndice 3

Código fuente del PDPACAD9

Este apéndice contiene el código fuente documentado del sistema **PDPACAD9**.

Figura

```
#include <iostream>
#include <qgl.h>
#include "ProGeo.h"
#include "AL.h"
#include <math.h>

//Elementos para crear un nodo Punto 3D
typedef float Punto3D[3];
struct node{
    float Punto3D[3];
    node *siguiente;// Pointer to next node
};

//Estructuras y funciones para realizar el conteo de operaciones
struct operaciones{
    int multiplicaciones;
    int sumas;
    int floats;
};
operaciones operator+(const operaciones op1, const operaciones op2);
void setOperacionesCeros(operaciones &op);

class Figura;

typedef node* apunt_nodo;

/*
    Clase lista de puntos.
    Implmenta una lista ligada con elementos punto 3D
    Permite Insertar, Eliminar y recorrer la lista
    para ir generando una figura en OpenGL
*/

class ListaPuntos{
private:
    apunt_nodo cabeza;
```

```

        apunt_nodo actual;
public:
    ListaPuntos();
    ~ListaPuntos();
    void InsertarNodo(float *V);
    void InsertarNodoPrincipio(float *V);
    void EliminarUltimoNodo();
    //void RecorrerLista();
    void MoverActual();
    void RetrocederActual();
    apunt_nodo getActual();
    friend class Figura;
};

/*
    LA clase figura.
    Contiene la lista de puntos que genera la figura y
    las características de la figura
*/

class Figura{
private:
    Punto3D *aux; //variable auxiliar de punto 3D para realizar operacines
    float color[3]; //Color de la figura en RGB
    float Pos[3];
    void NormalizaVector(Punto3D v); // Funcion para normalizar vector
    void Resta(Punto3D x,Punto3D y,Punto3D res); //Resta de elementos
Punto 3D
    void ProductoVectorial(Punto3D a,Punto3D b,Punto3D res);
    bool seleccion; //variable que indica si la figura esta seleccionada
public:
    void setPos(float x,float y,float z);
    float TE; //Indica si una figura esta extruida
    int indice; //Indice de la figura en el Dibujo
    void setPlano(int eje); //Plano en el que se dibujo la Figura
    int tipo; // Indica que tipo de figura es, cuadrado, circulo o poligono.
    void Extruir(float d,int tipo); //Funcion que extruye la figura
    ListaPuntos *lista; // Lista con los puntos de la figura
    Figura(); // Funcion Constructora
    void RecorrerLista(); // Dibuja la figura en pantalla, con funciones de
OpenGL
        /*
reflexion simple
            ReflexionSimpleAL, construye una figura aplicando la
            con algebra lineal.
            float  $\hat{i}$ , f, construyen el vector normal al plano de reflexion
            plano indica el plano donde se encuentra la figura
            indice, le asigna indice a la figura
            op registra las operaciones realizadas
        */

```



```

    Figura ReflexionSimpleAL(float *i,float *f,float *plano,int indice,operaciones
&op);
        /*
simple          RotacionSimpleAL, construye una figura aplicando la rotacion
               con algebra lineal.
               float i, f, construyen el vector normal al plano de reflexion
               angulo, angulo de rotacion
               indice, le asigna indice a la figura
               op registra las operaciones realizadas
        */
    Figura RotacionSimpleAL(float *i,float *f,float angulo,int indice,operaciones
&op);
        /*
reflexion simple ReflexionSimpleAG, construye una figura aplicando la
                 con algebra geometrica.
                 float i, f, construyen el vector normal al plano de reflexion
                 plano indica el plano donde se encuentra la figura
                 indice, le asigna indice a la figura
                 op registra las operaciones realizadas
        */
    Figura ReflexionSimpleAG(float *i,float *f,float *plano,int indice,operaciones
&op);
        /*
simple          RotacionSimpleAL, construye una figura aplicando la rotacion
               con algebra lineal.
               float i, f, construyen el vector normal al plano de reflexion
               angulo, angulo de rotacion
               indice, le asigna indice a la figura
               op registra las operaciones realizadas
        */
    Figura RotacionSimpleAG(float *i,float *f,float angulo,int indice,operaciones
&op);
    void setColor(float r,float g,float b); //Asigna el color
};

```

GLBox

```

#ifndef GLBOX_H
#define GLBOX_H
#define Frac_Circ 100 // Cien fracciones de circulo
#define PI 3.1415926535897932

#include <qgl.h>
#include "iostream"
#include "Figura.h"
#include <math.h>

```

```

#include <list>
#include "panel_resultados.h"

using namespace std;

/*
    Clase GLBox
    Pantalla realizada con OpenGL, despliega, las figuras
    permite dibujar en pantalla, y manipular la visualizacion
    grafica
*/

class GLBox : public QGLWidget{

    Q_OBJECT

public:

    GLBox( QWidget* parent, const char* name );
    ~GLBox();
    void    setF(int i); // Funcion que establece la operacion a realizar con la variable
F
    void    lzoom(); //Incrementa el Zoom
    void    Dzoom(); //Decrementa el Zoom
    void    Plano(int seleccion); // Establece el plano para dibujar
    void    Extruir(int tipo,float tam);/*1:afuera,2:adentro,3:ambos*/
    Figura  *lineas; // Variable auxiliar para dibujar tiras de lineas
    Figura  *f; // Variable que contiene una figura para luego ser insertada la la lista
    list<Figura> *ListaFiguras; // Lista de Figuras que seran mostradas en la pantalla
    int plano; // Indica el plano en el que se trabaja, 1-XY, 2 - XZ, 3-YZ
    panel_resultados* p_resultados; // Panel de resultados de las operaciones
    void RSAG(float *i,float *f,float *plano,int indice); /*REFLEXION SIMPLE CON
ALGEBRA GEOMETRICA*/
    void RoSAG(float *i,float *f,float angulo,int indice); /*ROTACION SIMPLE CON
ALGEBRA GEOMETRICA*/
    void RSAL(float *i,float *f,float *plano,int indice); /*REFLEXION SIMPLE CON
ALGEBRA LINEAL*/
    void RoSAL(float *i,float *f,float angulo, int indice); /*ROTACION SIMPLE CON
ALGEBRA LINEAL*/s
    Vector RotarCamara(Vector eje,Vector camara,float angulo);
    void  EstablecerColor(float r,float g,float b); // Establece el color para la siguiente
figura a ser dibujada
    float  AnguloRotacion; // Indica el angulo que ha rotado la camara
    void  setHerramienta(bool h); // Establece la herramienta a ser usada
    Vector ejeX; //
    Vector ejeY; //
    void  MoverSobreY(float angulo); //Mueve la camara sobre el eje Y
    void  MoverSobreX(float angulo); //Mueve la camara sobre el eje X
    void  Nuevo(); // Crea un nuevo Dibujo
    void  LimpiarOperaciones(); //Limpia el panel de resultados
    int   selID; //variable que indica el numero de figuras selccionadas

```

public slots:

```
    void mousePressEvent( QMouseEvent* event ); //Evento al presionar el boton del
mouse
    void mouseReleaseEvent( QMouseEvent* event ); // Evento al soltar el boton del
mouse
    void mouseMoveEvent( QMouseEvent* event ); // Evento al mover el mouse
```

protected:

```
// Funciones de configuracion de OpenGL
    void initializeGL();
    void paintGL();
    void Escena(GLenum mode);
    void resizeGL( int w, int h );
    void Coordinadas(GLfloat &x,GLfloat &y);
```

private:

```
    //Vector que indica la posicion de la camaras
    Vector camara;
    Vector auxX;
    Vector auxY;
```

```
    bool herramienta; /*falso: algebra lineal, verdadero: algebra geometrica*/
```

```
    //Variable auxiliares para dibujar una figura en un plano
    GLuint object;
    GLfloat xRot, yRot, zRot, scale, xTrans, yTrans,ix,iy,fx,fy;
    QPoint lastPos;
    QPoint inicio;
    QPoint final;
```

```
    int figuras; // Numero de figuras dibujadas
    void Inicializa_luz(); // Funcion que inicia la luz en OpenGL
```

```
    //Variables para la selccion con el mouse
    void anallizename( int name);
    void list_hits(GLint hits, GLuint buffer[]);
```

```
    int F; //Variable que indica la operacion que esta activada
    bool banderalinea; //Indica al sistema que se esta dibujando una tira de lineas
    int puntosEspecial; //
    float color[3]; //Indica el color que se asigna a las figuras al ser dibujadas
    int anguloX; //Angulo de rotacion en el eje X
    int anguloY; //Angulo de rotacion en el eje Y
    int *seleccion; //Arreglo que indica las figuras seleccionadas
```

```

//Multivectores auxiliares para rotar las figuras y almacenarlas en la memoria

Multivector Rx;
Multivector Rlx;
Multivector Ry;
Multivector Rly;

operaciones op;
operaciones opaux;
};
#endif // GLBOX_H

class Multivector;
/**ELEMENTOS BASICOS DEL MULTIVECTOR***/
struct Escalar{
    float e0;
};

struct Vector{
    float e1;
    float e2;
    float e3;
};

struct Bivector{
    float e1e2;
    float e1e3;
    float e2e3;
};

struct Trivector{
    float e1e2e3;
};
/**FINALIZA ELEMENTOS BASICOS DEL MULTIVECTOR***/

/*CONVERSION DE TIPOS*/
Vector ToBasic(float x,float y,float z);
float *ToVector(Vector V);
float *ToPointer(float x,float y,float z);
Vector ProductoVectorialAG(Vector v1,Vector v2);
/*FINALIZA CONVERSION DE TIPOS*/

/*FUNCIONES AUXILIARES*/
Vector ProductoVectorial(float ix,float iy,float iz,float fx,float fy,float fz);
/*FINALIZA FUNCIONES AUXILIARES*/

/**SUMA Y RESTA DE ELEMENTOS BASICOS***/
Escalar operator +(const Escalar& E1,const Escalar& E2);
Escalar operator -(const Escalar& E1,const Escalar& E2);
Vector operator +(const Vector& V1,const Vector& V2);

```

```
Vector operator -(const Vector& V1,const Vector& V2);
Bivector operator +(const Bivector& B1,const Bivector& B2);
Bivector operator -(const Bivector& B1,const Bivector& B2);
Trivector operator +(const Trivector& T1,const Trivector& T2);
Trivector operator -(const Trivector& T1,const Trivector& T2);
/**FINALIZA SUMA Y RESTA DE ELEMENTOS BASICOA*****/
```

```
/**MAGNITUD DE ELEMENTOS BASICOS DE UN MULTIVECTOR***/
float Magnitud(Escalar E);
float Magnitud(Vector V);
float Magnitud(Bivector B);
float Magnitud(Trivector T);
/**FINALIZA MAGNITUD DE ELEMENTOS BASICOS DEL MULTIVECTOR***/
```

```
/**INVERSO DE ELEMENTOS BASICOS DE UN MULTIVECTOR***/
Vector Inverso(Vector V);
Bivector Reverso(Bivector B);
Trivector Reverso(Trivector T);
/**FINALIZA INVERSO DE ELEMENTOS BASICOS DE UN MULTIVECTOR****/
```

```
/**PRODUCTO INTERIOR*****
Escalar ProductoInterior(Vector V1,Vector V2);
Vector ProductoInterior(Vector V,Bivector B);
Vector ProductoInterior(Bivector B,Vector V);
Bivector ProductoInterior(Vector V,Trivector T);
Escalar ProductoInterior(Bivector B1,Bivector B2);
Vector ProductoInterior(Bivector B,Trivector T);
/**FINALIZA PRODUCTO INTERIOR*****
```

```
/**Producto Exterior*****
Escalar ProductoExterior(Escalar E1,Escalar E2);
Vector ProductoExterior(Escalar E,Vector V);
Bivector ProductoExterior(Escalar E,Bivector B);
Trivector ProductoExterior(Escalar E,Trivector T);
Bivector ProductoExterior(Vector V1,Vector V2);
Trivector ProductoExterior(Vector V,Bivector B);
/**Finaliza producto exterior*****
```

```
Multivector operator +(const Multivector& m1,const Multivector& m2);
Multivector operator -(const Multivector& m1,const Multivector& m2);
Multivector ProductoGeometrico(Vector V1,Vector V2);
Multivector ProductoGeometrico(Multivector M, Vector V);
Multivector ProductoGeometrico(Multivector M1,Multivector M2);
    //friend Multivector
Vector ReflexionSimple(Vector X,Vector Y,Vector XI);
Multivector GeneraRotor(float angulo,Vector V);
Vector RotacionSimple(Multivector R, Vector V,Multivector RI);
Multivector Reverso(Multivector M);
```

```

/*
    Clase Multivector
    Aplica el algebra geometrica en 3D
*/

class Multivector{

public:
    Escalar E;
    Vector V;
    Bivector B;
    Trivector T;

    Multivector();
    Multivector(Escalar E,Vector V,Bivector B,Trivector T);
    friend Multivector operator +(const Multivector& m1,const Multivector& m2);
    friend Multivector operator -(const Multivector& m1,const Multivector& m2);
    friend Multivector ProductoGeometrico(Vector V1,Vector V2);
    friend Multivector ProductoGeometrico(Multivector M, Vector V);
    friend Multivector ProductoGeometrico(Multivector M1,Multivector M2);
    //friend Multivector
    friend Vector ReflexionSimple(Vector X,Vector Y,Vector XI);
    friend Multivector GeneraRotor(float angulo,Vector V);
    friend Vector RotacionSimple(Multivector R, Vector V,Multivector RI);
    friend Multivector Reverso(Multivector M);

};

```

Clase matriz

```
/**ELEMENTOS BASICOS DE MATRIZ**/
```

```
struct vector{ //estructura para el manejo de vectores, compuesta de tres float para manejar un  
    float v1; //vector en 3D  
    float v2;  
    float v3;
```

```
};
```

```
/**FINALIZA ELEMENTOS BASICOS DE MATRIZ**/
```

```
/**FUNCIONES AUXILIARES**/
```

```
vector multiplicacionVE(float escalar, vector y);//multiplica un escalar con un vector
```

```
vector restaVector(vector x, vector y);//resta dos vectores
```

```
vector sumaVector(vector x, vector y);//suma dos vectores
```

```
void multiplicacionmatrixvector(float matrixdereflexion1[3][3],float v[3],float rotacion[3]); // multiplica  
una matriz 3x3 con un vector de dimensión 3.
```

```
void construyematrixrotacion(float eje[3],float M[3][3],float angulo); //construye la matriz de rotación a  
partir de los básicos e1,e2 y e3, matriz que rotara determinado ángulo
```

```
/**FINALIZA FUNCIONES AUXILIARES**/
```

```
/**PRODUCTO PUNTO**/
```

```
float productoPunto(vector N, vector V);//realiza el producto punto de dos vectores
```

```
/**FINALIZA PRODUCTO PUNTO**/
```

```
/**PRODUCTO CRUZ**/
```

```
vector ProductoVectorialAL(vector v1,vector v2); realiza el producto cruz entre dos vectores
```

```
/**FINALIZA PRODUCTO CRUZ**/
```

```
/**REFLEXION SIMPLE**/
```

```
vector reflexionSimple(float matrixdereflexion[3][3], vector V);//realiza la reflexión del vector V sobre la  
normal N
```

```
/**FINALIZA REFLEXION SIMPLE**/
```

```
/**ROTACION SIMPLE**/
```

```
vector rotacionSimple(float matrixderotacion[3][3], vector V);//realiza la rotación simple del vector V  
multiplicándolo por la matriz matrixderotacion
```

```
/**FINALIZA ROTACION SIMPLE**/
```