MATRIZES DE ROTAÇÃO A PARTIR DE QUATÉRNIONS E A CRIAÇÃO DE UM COMPONENTE C++ BUILDER PARA VISUALIZAÇÃO DE FUNÇÕES MATEMÁTICAS NO R^3

Lincoln C. Zamboni*

Sergio V. D. Pamboukian**

Resumo

A cinemática de imagens vetoriais (rotação e translação) na computação gráfica é concebida sob as mais variadas óticas dos modelos matemáticos até então propostos.

Tais modelos matemáticos formalizam uma base sólida para a concepção de algoritmos eficientes quando observados pela função "O" de Paul Bachmann.

Este artigo mostra dois modelos matemáticos no estudo da cinemática de imagens vetoriais: os quatérnions e as matrizes de rotação.

Optou-se por implementar em C++ um algoritmo usando matrizes de rotação por três simples razões:

1. didática-pedagógica, uma vez que os alunos iniciantes em ciências exatas nas universidades já possuem familiaridade com matrizes;
2. multidisciplinaridade almejada entre Álgebra Linear, Geometria Analítica, Mecânica Vetorial, Cálculo Diferencial e Integral e Computação, dentre tantas outras;

* Universidade Presbiteriana Mackenzie
E-mail: lincoln.zamboni@mackenzie.com.br

** Universidade Presbiteriana Mackenzie
E-mail: sergiop@mackenzie.com.br

3. PRIMA (PRojeto de Integração Multidisciplinar Acadêmico) já proposto e desenvolvido pelos alunos da graduação a cada semestre.

Foi utilizada a Programação Orientada a Objeto (POO) em detrimento à Programação Estruturada (PE) para trazer ao aluno conceitos de como criar e reutilizar um componente no ambiente do C++ Builder. Um componente *TImage3D* foi criado e reutilizado em uma aplicação que desenha gráficos de funções de duas variáveis.

Abstract

The cinematic of vectorial images (rotation and translocation) in computer graphics is conceived through different ways of mathematical model views so far proposed.

These mathematical models build a solid base in order to create efficient algorithms under the scope of the "O" Bachmann's function.

This article focuses on two mathematical models applied on the cinematic of the vectorial images: quaternions and rotation matrixes.

There were three reasons to choose on implementing a C++ algorithm using rotation matrixes:

1. methodology, because the beginners in sciences at universities have good knowledge about matrixes;
2. multidisciplinarity among Linear Algebra, Analytic Geometry, Vectorial Mechanics, Calculus and Computer Science;
3. PRIMA "PRojeto de Integração Multidisciplinar Acadêmico", has already been proposed and developed by students in each semester.

Object Oriented Programming (OOP) has been chosen under Structured Programming (SP) to introduce students into some concepts, such as creating and reusing a component in a C++ Builder environment. A component named TImage3D has been created and reused in an application which draws functions of two variables.

1 INTRODUÇÃO HISTÓRICA

O caráter internacional da álgebra abstrata do século XIX foi percebido pelo fato de as duas contribuições mais revolucionárias terem sido feitas, em 1843, por Willian Rowan Hamilton (1805 – 1865) e, em 1847, por George Boole (1815 – 1864).

O pai de Hamilton era advogado e, assim como sua mãe, dotado de um intelecto acima da média. Morreram quando Hamilton ainda era criança. A educação de Hamilton foi garantida por um tio lingüista, versado no estudo das línguas. Aos 5 anos de idade, lia grego, hebraico e latim. Um encontro com um calculista e sua amizade com Wordsworth e Coleridge estimularam o pequeno Willian ao gosto pela matemática. Ingressou no Trinity College, em Dublin, e, aos 22 anos, foi nomeado astrônomo real da Irlanda.

Um certo dia do ano de 1843, enquanto passeava com a esposa às margens do Royal Canal, obteve uma inspiração na solução dos problemas algébricos que repercutiam em sua mente brilhante. Nesse mesmo ano, submeteu à Real Academia Irlandesa sua teoria segundo a qual a palavra quatérnion foi empregada para denotar certo quadrinômio, que constitui um corpo, exceto pela comutatividade da multiplicação, e que se pode representar pela soma $r+v_x \cdot i+v_y \cdot j+v_z \cdot k$, em que r, v_x, v_y e v_z são reais. O primeiro dos termos do quatérnion foi chamado de parte real, enquanto os outros três, em conjunto, de parte imaginária por analogia com as terminologias da álgebra ordinária.

2 RESUMO ALGÉBRICO

A representação algébrica de um quatérnion é $q = r+v$, em que $r=r \cdot 1$ é chamada de parte real e $v=v_x \cdot i+v_y \cdot j+v_z \cdot k$ é chamada de parte imaginária.

O conjugado de um quatérnion $q=r+v$ é definido por $\overline{q} \equiv r-v$. Note que a parte imaginária tem o sinal invertido.

O módulo de um quatérnion $q=r+v$ é definido por $|q| \equiv \sqrt{r^2 + v^2}$, em que $r^2=r \cdot r$ é o produto entre o escalar r e ele mesmo e $v^2=v \cdot v$ é o produto escalar do vetor v por ele mesmo.

A soma entre dois quatérnions $q_1=r_1+v_1$ e $q_2=r_2+v_2$ é definida por $q_1+q_2 \equiv (r_1+r_2) +(v_1+v_2)$, em que r_1+r_2 é a soma de dois escalares (parte real do primeiro quatérnion

com a parte real do segundo quatérnion) e v_1+v_2 é a soma de dois vetores (parte imaginária do primeiro quatérnion com a parte imaginária do segundo quatérnion).

O produto entre dois quatérnions $q_1=r_1+v_1$ e $q_2=r_2+v_2$ é definido por $q_1 \cdot q_2 \equiv (r_1 \cdot r_2 - v_1 \cdot v_2)+(r_1 \cdot v_2 + v_1 \cdot r_2 + v_1 \times v_2)$, em que $r_1 \cdot r_2 - v_1 \cdot v_2$ é a diferença entre dois escalares (o primeiro vem de um produto entre dois escalares $r_1 \cdot r_2$ e o segundo de um produto escalar entre dois vetores $v_1 \cdot v_2$) e $r_1 \cdot v_2 + v_1 \cdot r_2 + v_1 \times v_2$ é a soma entre três vetores (o primeiro é um produto escalar $r_1 \cdot v_2$, o segundo também é um produto escalar $v_1 \cdot r_2$ e o terceiro é um produto vetorial $v_1 \times v_2$).

A Tabela 1 ilustra os valores da base canônica utilizados na multiplicação de dois quatérnions $q_1=r_1+v_{1x} \cdot i+v_{1y} \cdot j+v_{1z} \cdot k$ e $q_2=r_2+v_{2x} \cdot i+v_{2y} \cdot j+v_{2z} \cdot k$. O produto não é comutativo e com a ajuda da tabela obtém-se:

$$q_1 \cdot q_2 = (r_1 \cdot r_2 - v_{1x} \cdot v_{2x} - v_{1y} \cdot v_{2y} - v_{1z} \cdot v_{2z}) + (r_1 \cdot v_{2x} + v_{1x} \cdot r_2 + v_{1y} \cdot v_{2z} - v_{1z} \cdot v_{2y}) \cdot i + (r_1 \cdot v_{2y} - v_{1x} \cdot v_{2z} + v_{1y} \cdot r_2 + v_{1z} \cdot v_{2x}) \cdot j + (r_1 \cdot v_{2z} + v_{1x} \cdot v_{2y} - v_{1y} \cdot v_{2x} + v_{1z} \cdot r_2) \cdot k$$

A Figura 1 ilustra os valores utilizados no produto vetorial entre dois vetores $v_1=v_{1x} \cdot i+v_{1y} \cdot j+v_{1z} \cdot k$ e $v_2=v_{2x} \cdot i+v_{2y} \cdot j+v_{2z} \cdot k$. Esse produto é uma das parcelas constituintes da parte imaginária do produto de dois quatérnions. Um exemplo de uso dessa figura é $k \times j=-i$, pois caminha-se de k para j no sentido anti-horário (negativo) e encontra-se i.

TABELA 1

Valores úteis para a multiplicação de quatérnions

	1	i	j	k
1	1	i	j	k
i	i	-1	k	j
j	j	-k	-1	i
k	k	j	-i	-1

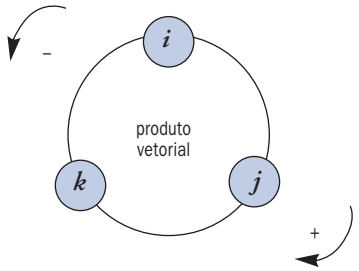

Figura 1 Lembrete para efetuar o produto vetorial

Um relação importante é $q \cdot \overline{q} = (r+v) \cdot (r-v) = r^2+v^2 = (r-v) \cdot (r+v) = \overline{q} \cdot q = |q|^2$. Logo $q \cdot \overline{q} = \overline{q} \cdot q = |q|^2$, isto é, o produto entre um quatérnion e seu conjugado é comutativo e resulta no módulo do quatérnion ao quadrado.

A diferença entre dois quatérnions $q_1 = r_1 + v_1$ e $q_2 = r_2 + v_2$ é definida por $q_1 - q_2 \equiv (r_1 - r_2) + (v_1 - v_2)$ e a divisão por $\frac{q_1}{q_2} \equiv \frac{q_1 \cdot \overline{q}_2}{q_2 \cdot \overline{q}_2} = \left(\frac{r_1 \cdot r_2 + v_1 \cdot v_2}{|q_2|^2} \right) + \left(\frac{v_1 \cdot r_2 - r_1 \cdot v_2 - v_1 \times v_2}{|q_2|^2} \right)$, em que $q_2 \neq 0$.

As operações de soma e multiplicação com as propriedades que se seguem conferem ao conjunto dos quatérnions uma estrutura de anel divisão:

1) $\forall x,y \in Q: x+y \in Q$ fechamento em relação a soma);
2) $\forall x,y,z \in Q: (x+y)+z = x+(y+z)$ (associativa da soma);
3) $\forall x,y \in Q: x+y = y+x$ (comutativa da soma);
4) $\exists 0 \in Q, \forall x \in Q: 0+x = x+0 = x$ (existência do elemento neutro);
5) $\exists -x \in Q, \forall x \in Q: (-x)+x = x+(-x) = 0$ (existência do elemento inverso aditivo);
6) $\forall x,y \in Q: x \cdot y \in Q$ (fechamento em relação ao produto);
7) $\forall x,y,z \in Q: (x \cdot y) \cdot z = x \cdot (y \cdot z)$ (associativa do produto);
8) $\exists 1 \in Q, \forall x \in Q: 1 \cdot x = x \cdot 1 = x$ (existência do elemento unidade);
9) $\exists 1/x \in Q, \forall x \in Q - \{0\}: (1/x) \cdot x = x \cdot (1/x) = 1$ (existência do elemento inverso multiplicativo);
10) $\forall x,y,z \in Q: x \cdot (y+z) = x \cdot y + x \cdot z$ (distributiva a esquerda);
11) $\forall x,y,z \in Q: (x+y) \cdot z = x \cdot z + y \cdot z$ (distributiva a direita).

Sob a ótica algébrica, tanto faz considerar os quatérnions uma quádrupla ordenada ou um escalar somado com um vetor ou outra concepção similar. Isso é garantido quando existe um isomorfismo entre tais conjuntos, considerando as operações de soma e produto, isto é, existe uma bijeção *iso* entre tais conjuntos, em que $iso(x+y) = iso(x) + iso(y)$ e $iso(x \cdot y) = iso(x) \cdot iso(y)$.

3 GEOMETRIA DA ROTAÇÃO NO ESPAÇO

Observando a Figura 2 percebe-se que $P_r = P + R$ e $|u \times P| = |P| \cdot \text{sen}\ \alpha$.

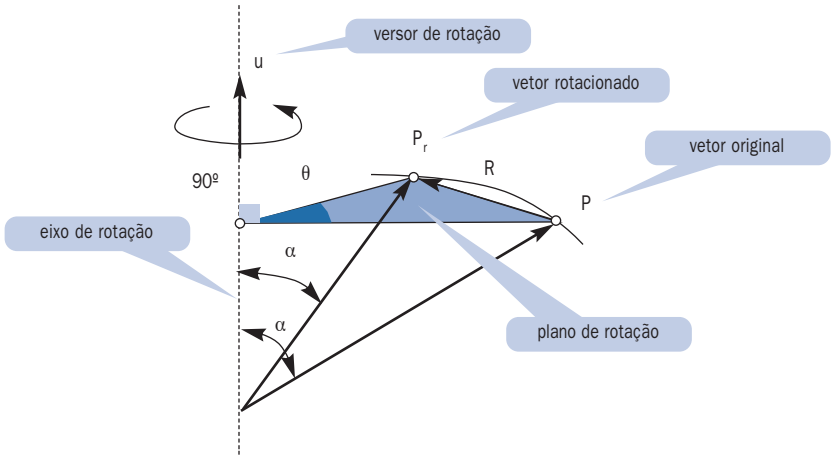

Figura 2 Rotação usando vetores

Observando a Figura 3, percebe-se que $|R| = 2 \cdot |P| \cdot \text{sen}\ \alpha \cdot \cos(90^\circ - \frac{\theta}{2}) =$

$= 2 \cdot |u \times P| \cdot \text{sen}\ \frac{\theta}{2} \cdot$

Assim.

$$R = |R| \cdot \left[\cos \frac{\theta}{2} \cdot t + \text{sen}\ \frac{\theta}{2} \cdot n \right] = 2 \cdot |u \times P| \cdot \text{sen}\ \frac{\theta}{2} \cdot \left[\cos \frac{\theta}{2} \cdot \frac{u \times P}{|u \times P|} + \text{sen}\ \frac{\theta}{2} \cdot \left(u \times \frac{u \times P}{|u \times P|} \right) \right] =$$

$$= 2 \cdot \text{sen}\ \frac{\theta}{2} \cdot \left[\cos \frac{\theta}{2} \cdot (u \times P) + \text{sen}\ \frac{\theta}{2} \cdot u \times (u \times P) \right].$$

Finalizando:

$P_r = P + R$, em que $R = 2 \cdot \text{sen}\ \frac{\theta}{2} \cdot \left[\cos \frac{\theta}{2} \cdot (u \times P) + \text{sen}\ \frac{\theta}{2} \cdot u \times (u \times P) \right]$.

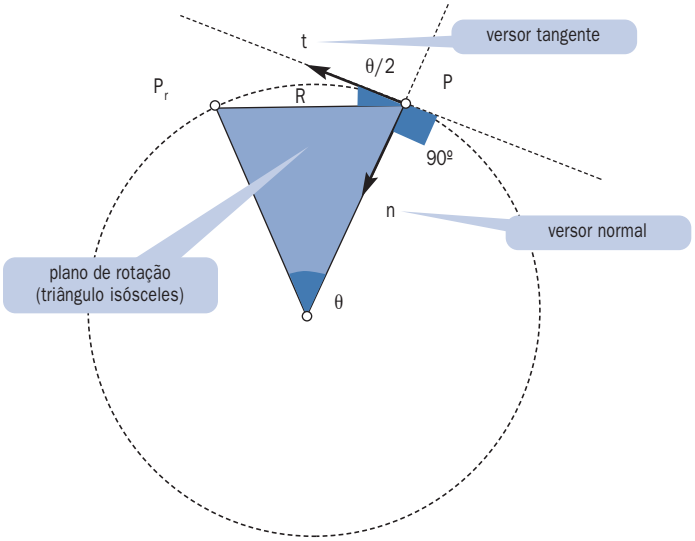

Figura 3 Visão no plano de rotação

4 ROTAÇÃO NO ESPAÇO COM QUATÉRNIONS

Uma identidade importante pode ser observada por:

$$\text{sen}^2\frac{\theta}{2}\cdot u\times(u\times P)=\text{sen}^2\frac{\theta}{2}\cdot\left[(u\cdot P)\cdot u-(u\cdot u)\cdot P\right]=\text{sen}^2\frac{\theta}{2}\cdot(u\cdot P)\cdot u-\text{sen}^2\frac{\theta}{2}\cdot P=$$

$$=\text{sen}^2\frac{\theta}{2}\cdot(u\cdot P)\cdot u-(1-\cos^2\frac{\theta}{2})\cdot P=\text{sen}^2\frac{\theta}{2}\cdot(u\cdot P)\cdot u+\cos^2\frac{\theta}{2}\cdot P-P.$$

Reescrevendo, tem-se $\text{sen}^2\frac{\theta}{2}\cdot u\times(u\times P)+P=\text{sen}^2\frac{\theta}{2}\cdot(u\cdot P)\cdot u+\cos^2\frac{\theta}{2}\cdot P.$

Utilizando essa identidade e as equações finalizadas no item 3, encontra-se:

$$P_r = P + R = P + 2 \cdot \operatorname{sen}\frac{\theta}{2} \cdot \left[\cos\frac{\theta}{2} \cdot (u \times P) + \operatorname{sen}\frac{\theta}{2} \cdot u \times (u \times P) \right] =$$

$$= P + 2 \cdot \operatorname{sen}^2 \frac{\theta}{2} \cdot u \times (u \times P) + 2 \cdot \operatorname{sen}\frac{\theta}{2} \cdot \cos\frac{\theta}{2} \cdot (u \times P) = -\operatorname{sen}\frac{\theta}{2} \cdot \cos\frac{\theta}{2} \cdot (u \cdot P) +$$

$$\cos\frac{\theta}{2} \cdot \operatorname{sen}\frac{\theta}{2} \cdot (P \cdot u) + \operatorname{sen}^2 \frac{\theta}{2} \cdot (u \times P) \cdot u + \operatorname{sen}^2 \frac{\theta}{2} \cdot (u \cdot P) \cdot u + \cos^2 \frac{\theta}{2} \cdot P +$$

$$\cos\frac{\theta}{2} \cdot \operatorname{sen}\frac{\theta}{2} \cdot (u \times P) - \cos\frac{\theta}{2} \cdot \operatorname{sen}\frac{\theta}{2} \cdot (P \times u) - \operatorname{sen}^2 \frac{\theta}{2} \cdot (u \times P) \cdot u =$$

$$= \left(-\operatorname{sen}\frac{\theta}{2} \cdot (u \cdot P) + \cos\frac{\theta}{2} \cdot P + \operatorname{sen}\frac{\theta}{2} \cdot (u \times P) \right) \cdot \left(\cos\frac{\theta}{2} - \operatorname{sen}\frac{\theta}{2} \cdot u \right) =$$

$$= \underbrace{\left(\cos\frac{\theta}{2} + \operatorname{sen}\frac{\theta}{2} \cdot u \right)}_{q} \cdot (0 + P) \cdot \underbrace{\left(\cos\frac{\theta}{2} - \operatorname{sen}\frac{\theta}{2} \cdot u \right)}_{\overline{q}} = q \cdot P \cdot \overline{q} \,.$$

Reescrevendo, tem-se $P_r = P + R = q \cdot P \cdot \overline{q}$, em que:

$$R = 2 \cdot \operatorname{sen}\frac{\theta}{2} \cdot \left[\cos\frac{\theta}{2} \cdot (u \times P) + \operatorname{sen}\frac{\theta}{2} \cdot u \times (u \times P) \right] \text{ e } q = \left(\cos\frac{\theta}{2} + \operatorname{sen}\frac{\theta}{2} \cdot u \right).$$

5 MATRIZES DE ROTAÇÃO

A posição de um ponto no espaço pode ser definida de várias maneiras. A mais comum é com o uso do sistema de coordenadas cartesianas, onde um ponto é posicionado através de três coordenadas reais (*x, y, z*).

Para visualizar uma função em 3D na tela do computador (que é plana – 2D), pode-se projetar cada ponto (*x, y, z*) da função no plano definido pela tela, usando, para tanto, uma projeção cônica ou cilíndrica. Os itens seguintes ilustram uma projeção cilíndrica ortogonal ao plano definido pela tela.

5.1 Caso particular: rotação em torno do eixo *z*

Considerando apenas a rotação em torno do eixo *z* de um ângulo ω_z, conforme ilustrado na Figura 4, tem-se que, a partir da igualdade vetorial $x \cdot i + y \cdot j + z \cdot k = x' \cdot i' + y' \cdot j' + z' \cdot k'$ (i, j, k, i', j' e k' são os versores dos eixos x, y, z, x', y' e z', respectivamente), multiplicando escalarmente ambos os membros por i (o produto escalar entre dois versores é o co-seno do ângulo entre eles), encontra-se: $x = x' \cdot (i' \cdot i) + y' \cdot (j' \cdot i) = x' \cdot \cos\omega_z + y' \cdot \cos(90 - \omega_z) = = x' \cdot \cos\omega_z - y' \cdot \text{sen}\omega_z$).

Analogamente, $y = x' \cdot \text{sen}\omega_z + y' \cdot \cos\omega_z$ e $z = z'$.

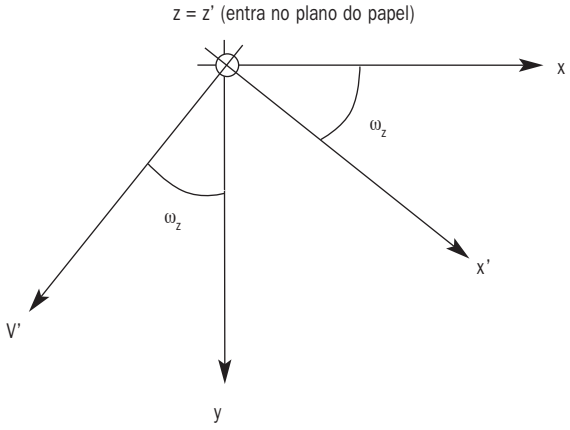

Figura 4 Rotação em torno do eixo z

Matricialmente tem-se $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\omega_z & -\text{sen}\omega_z & 0 \\ \text{sen}\omega_z & \cos\omega_z & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotação em torno de z } (R_{xy})} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$.

5.2 Caso genérico: rotação e translação

Considerando as três rotações possíveis e também as três translações, tem-se a seguinte equação matricial:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \underbrace{\begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}}_{\text{translação}} + \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\omega_x & -\operatorname{sen}\omega_x \\ 0 & \operatorname{sen}\omega_x & \cos\omega_x \end{bmatrix}}_{\text{rotação em torno de x } (R_{yz})} \cdot \underbrace{\begin{bmatrix} \cos\omega_y & 0 & \operatorname{sen}\omega_y \\ 0 & 1 & 0 \\ -\operatorname{sen}\omega_y & 0 & \cos\omega_y \end{bmatrix}}_{\text{rotação em torno de y } (R_{xz})} \cdot \underbrace{\begin{bmatrix} \cos\omega_z & -\operatorname{sen}\omega_z & 0 \\ \operatorname{sen}\omega_z & \cos\omega_z & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{rotação em torno de z } (R_{xy})} \cdot \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

Considerando, conforme ilustrado na Figura 5, apenas translações nas direções dos eixos x e y e tomando a projeção no plano xy (despreza-se z), encontra-se, a partir da equação matricial anterior:

$$\begin{cases} x = x_t + (cy \cdot cz) \cdot x' - (cy \cdot sz) \cdot y' + sy \cdot z' \\ y = y_t + (a \cdot cz + cx \cdot sz) \cdot x' + (-a \cdot sz + cx \cdot cz) \cdot y' - (sx \cdot cy) \cdot z' \end{cases}$$, em que:

$cx=\cos\omega_x$, $cy=\cos\omega_y$, $cz=\cos\omega_z$, $sx=\operatorname{sen}\omega_x$, $sy=\operatorname{sen}\omega_y$, $sz=\operatorname{sen}\omega_z$ e $a=sx \cdot sy$.

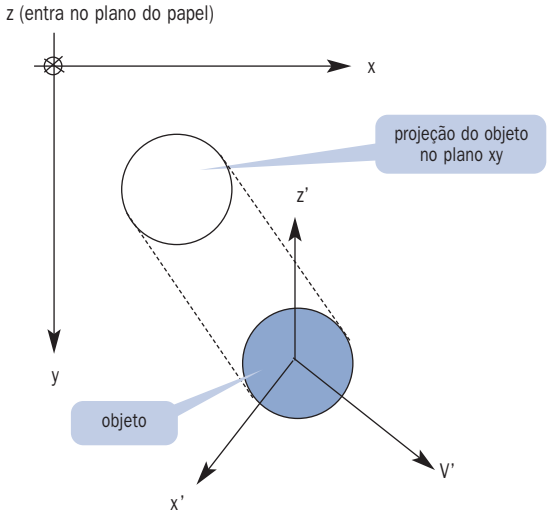

Figura 5 Projeção no plano *xy*

6 MATRIZES DE ROTAÇÃO A PARTIR DE QUATÉRNIONS

Considerando os quatérnions $q_1=r_1+v_1=r_1+v_{1x}\cdot i+v_{1y}\cdot j+v_{1z}\cdot k$ e $q_2=r_2+v_2=r_2+v_{2x}\cdot i+v_{2y}\cdot j+v_{2z}\cdot k$ juntamente com seu produto, encontra-se:

$$q_1\cdot q_2=(r_1\cdot r_2-v_{1x}\cdot v_{2x}-v_{1y}\cdot v_{2y}-v_{1z}\cdot v_{2z})+(r_1\cdot v_{2x}+v_{1x}\cdot r_2+v_{1y}\cdot v_{2z}-v_{1z}\cdot v_{2y})+$$
$$(r_1\cdot v_{2y}-v_{1x}\cdot v_{2z}+v_{1y}\cdot r_2+v_{1z}\cdot v_{2x})+(r_1\cdot v_{2z}+v_{1x}\cdot v_{2y}-v_{1y}\cdot v_{2x}+v_{1z}\cdot r_2)=$$

$$=\left(\underbrace{\begin{bmatrix} r_2 & -v_{2x} & -v_{2y} & -v_{2z} \\ v_{2x} & r_2 & v_{2z} & -v_{2y} \\ v_{2y} & -v_{2z} & r_2 & v_{2x} \\ v_{2z} & v_{2y} & -v_{2x} & r_2 \end{bmatrix}}_{R(q_2)}\cdot\underbrace{\begin{bmatrix} r_1 \\ v_{1x} \\ v_{1y} \\ v_{1z} \end{bmatrix}}_{q_1^T}\right)^T=\left(\underbrace{\begin{bmatrix} r_1 & -v_{1x} & -v_{1y} & -v_{1z} \\ v_{1x} & r_1 & -v_{1z} & v_{1y} \\ v_{1y} & v_{1z} & r_1 & -v_{1x} \\ v_{1z} & -v_{1y} & v_{1x} & r_1 \end{bmatrix}}_{R(q_1)}\cdot\underbrace{\begin{bmatrix} r_2 \\ v_{2x} \\ v_{2y} \\ v_{2z} \end{bmatrix}}_{q_2^T}\right)^T$$

A partir do anteriormente encontrado e considerando $q=\left(\cos\frac{\theta}{2}+\text{sen}\,\frac{\theta}{2}\cdot u\right)$ e $\overline{q}=\left(\cos\frac{\theta}{2}-\text{sen}\,\frac{\theta}{2}\cdot u\right)$, tem-se:

$$P_r^T=(q\cdot P\cdot\overline{q})^T=(q\cdot(P\cdot\overline{q}))^T=\left(q\cdot\left(R(\overline{q})\cdot P^T\right)^T\right)^T=$$

$$\left(q\cdot\left(P\cdot R(\overline{q})^T\right)\right)^T=\left((q\cdot P)\cdot R(\overline{q})^T\right)^T=\left(\left(R(\overline{q})\cdot P^T\right)^T\cdot R(\overline{q})^T\right)^T=\left(P\cdot R(q)^T\cdot R(\overline{q})^T\right)^T=$$

$$=R(\overline{q})\cdot R(q)\cdot P^T.$$

Reescrevendo, $P_r^T=R(\overline{q})\cdot R(q)\cdot P^T$.

6.1 Rotação em torno do eixo *z*

Para efetuar uma rotação em torno do eixo z, valendo-se de quatérnions, utiliza-se $q = \left(\cos\frac{\omega_z}{2} + \text{sen}\,\frac{\omega_z}{2} \cdot k \right)$ e $\bar{q} = \left(\cos\frac{\omega_z}{2} - \text{sen}\,\frac{\omega_z}{2} \cdot k \right)$.

Assim,

$$R(\bar{q}) = \begin{bmatrix} \cos\frac{\omega_z}{2} & 0 & 0 & \text{sen}\,\frac{\omega_z}{2} \\ 0 & \cos\frac{\omega_z}{2} & -\text{sen}\,\frac{\omega_z}{2} & 0 \\ 0 & \text{sen}\,\frac{\omega_z}{2} & \cos\frac{\omega_z}{2} & 0 \\ -\text{sen}\,\frac{\omega_z}{2} & 0 & 0 & \cos\frac{\omega_z}{2} \end{bmatrix} \text{ e}$$

$$R(q) = \begin{bmatrix} \cos\frac{\omega_z}{2} & 0 & 0 & -\text{sen}\,\frac{\omega_z}{2} \\ 0 & \cos\frac{\omega_z}{2} & -\text{sen}\,\frac{\omega_z}{2} & 0 \\ 0 & \text{sen}\,\frac{\omega_z}{2} & \cos\frac{\omega_z}{2} & 0 \\ \text{sen}\,\frac{\omega_z}{2} & 0 & 0 & \cos\frac{\omega_z}{2} \end{bmatrix}.$$

Logo $\begin{bmatrix} 0 \\ x_{P_r} \\ y_{P_r} \\ z_{P_r} \end{bmatrix} = P_r^T = R(\bar{q}) \cdot R(q) \cdot P^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\omega_z & -\text{sen}\,\omega_z & 0 \\ 0 & \text{sen}\,\omega_z & \cos\omega_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ x_P \\ y_P \\ z_P \end{bmatrix}$.

Reescrevendo, $\begin{bmatrix} x_{P_r} \\ y_{P_r} \\ z_{P_r} \end{bmatrix} = \begin{bmatrix} \cos\omega_z & -\text{sen}\,\omega_z & 0 \\ \text{sen}\,\omega_z & \cos\omega_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_P \\ y_P \\ z_P \end{bmatrix}$, que é o mesmo resultado encontrado no item 5.1.

As rotações em torno dos outros dois eixos *x* e *y* são análogas.

7 IMPLEMENTAÇÃO DE MATRIZES DE ROTAÇÃO EM C++

Uma das características mais interessantes do *Borland* C++ *Builder* é permitir a criação de novos componentes ou a modificação de componentes que já existem. Dessa forma, é possível criar componentes específicos para aplicação ou customizar um componente já existente para atender necessidades.

Todo componente é implementado por meio de uma classe C++. Essa classe pode ser derivada e depois modificada, gerando novos componentes. A classe fundamental a partir da qual todos os outros componentes são gerados é a *TComponent*.

O componente *TImage3D*, apresentado neste trabalho, foi criado a partir do componente *TImage*. Foram acrescentados ao componente original propriedades e métodos que permitem a exibição de funções matemáticas de duas variáveis utilizando o conceito de matrizes de rotação.

8 CRIANDO UM COMPONENTE

A criação de um novo componente envolve quatro etapas: criar uma *Unit* em que será feita a implementação do componente, derivar o componente de outro já existente, declarar um novo construtor para a classe e registrar o componente.

8.1 Assistente para criação de componentes

A maneira mais fácil de se criar um componente é utilizar o assistente para criação de componentes (*Component Wizard*), que pode ser invocado por meio da opção New *Component* do menu *Component*. Como se vê na Figura 6, esse assistente permite que o programador escolha o nome da classe do novo componente, seu antecessor (componente do qual será derivado) e a guia em que será instalado, dentre outras coisas.

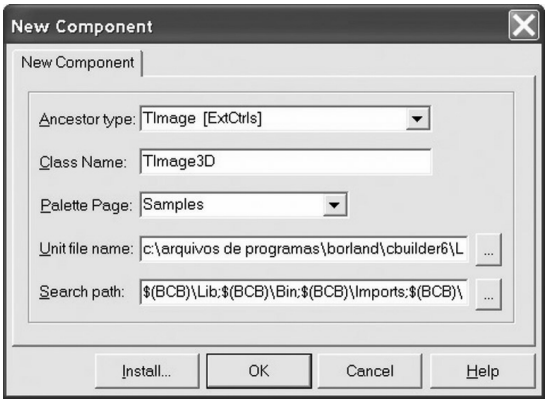

Figura 6 Assistente para criação de componente

O Quadro 1 mostra o arquivo de cabeçalho (.H) gerado pelo assistente, em que está definida a estrutura da nova classe.

QUADRO 1

Arquivo de cabeçalho Image3D.h

```
#ifndef Image3DH
#define Image3DH
//---------------------------------------------------
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <Controls.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------
class PACKAGE TImage3D : public TImage
{
private:
protected:
public:
 __fastcall TImage3D(TComponent* Owner);
__published:
};
//---------------------------------------------------
#endif
```

No Quadro 2, pode-se ver o arquivo fonte (.CPP) criado pelo assistente. Nele já estão definidos os códigos do construtor da nova classe e do registro do componente.

QUADRO 2

Arquivo fonte Image3D.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "Image3D.h"
#pragma package(smart_init)
//---------------------------------------------------
// ValidCtrCheck is used to assure that the components
// created do not have any pure virtual functions.
//
static inline void ValidCtrCheck(TImage3D *)
{
 new TImage3D(NULL);
}
//-----------CONSTRUTOR-----------------------------
__fastcall TImage3D::TImage3D(TComponent* Owner):
 TImage(Owner)
{
}
//------------REGISTRO-------------------------------
namespace Image3d1
{
 void __fastcall PACKAGE Register()
 {
  TComponentClass classes[1] = {__classid(TImage3D)};
  RegisterComponents("Samples", classes, 0);
 }
}
```

Uma vez criada a estrutura do novo componente, o próximo passo é definir suas propriedades, métodos e eventos.

8.2 Ajustando propriedades, métodos e eventos

Existem vários níveis de acesso para os dados, propriedades e métodos de uma classe:

- *private* – acessível apenas na classe onde foi definido;
- *protected* – acessível na classe onde foi definido e em seus descendentes;

- *public* – acessível a todo o código;
- *published* – acessível a todo o código, acessível a partir do *object*

8.2.1 Inspector *e valor armazenado no arquivo que descreve o formulário*

Para criar uma nova propriedade, é necessário especificar seu nome, tipo e os métodos utilizados para a leitura e escrita de seu valor. Se não houver método para a escrita, a propriedade é considerada como somente para leitura (*ReadOnly*).

No padrão empregado para nomear os métodos de cada classe, utiliza-se o prefixo *Set* quando o método é utilizado para alterar um dado da classe e o prefixo *Get* quando o método retorna o valor atual de um dado da classe. As variáveis que armazenam os valores das propriedades devem ter seus nomes iniciados pela letra "F" (inicial da palavra *Field* = Campo).

QUADRO 3

Definindo uma nova propriedade para o componente

```
class PACKAGE TMeuComponente : public TComponent

{
private:
 int FValor;
 int __fastcall GetValor();
 void __fastcall SetValor(int ACount);
public:
 __property int Valor = {read=GetValor, write=SetValor,
                         stored=true, default=10};
 ...
};
```

Como se vê no exemplo apresentado no Quadro 3, a propriedade Valor é do tipo *int* e possui dois métodos: *GetValor* utilizado para leitura (*read*) do valor da propriedade e o método *SetValor* utilizado para escrita (*write*) do valor. A opção *default*=10 indica o valor padrão desta propriedade. A opção *stored*=true indica que o valor da propriedade deve ser armazenado e salvo na estrutura do formulário, se for diferente do *default*.

Podem-se também dispensar os métodos de leitura e escrita e alterar diretamente o valor da variável interna (declarada na seção private) que armazena o valor da propriedade, indicando seu nome nas opções *read* e *write* (Quadro 4).

QUADRO 4

Definindo propriedade sem a utilização de métodos específicos para leitura e escrita

```
class PACKAGE TMeuComponente : public TComponent

{
private:
 int FValor;
public:
 __property int Valor = {read=FValor, write=FValor,
                          stored=true, default=10};
 ...
};
```

8.3 Registrando um componente

Depois de criado, o componente deve ser instalado e registrado. Isso pode ser feito por meio da opção *Install Component do* menu *Component.*

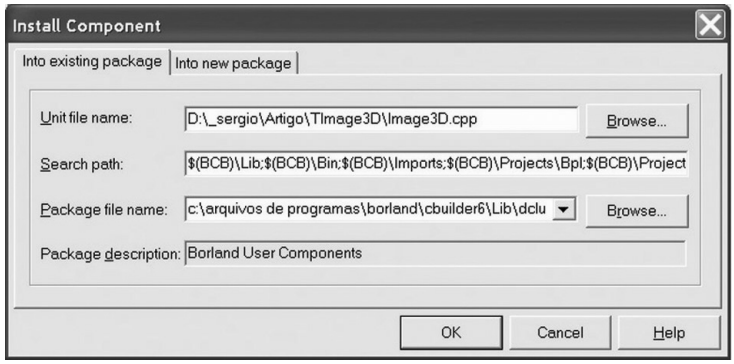

Figura 7 Instalando um novo componente

Nessa janela, deve-se especificar a *Unit* que contém a implementação do novo componente e o nome do pacote (*Package*) em que ela deverá ser instalada. Geralmente os componentes criados pelo usuário utilizam o pacote padrão *dclusr.bpk.*

Em seguida, será pedido que o usuário confirme a recompilação e a instalação do pacote. Após a compilação, será exibida outra mensagem indicando que o novo componente foi instalado e registrado. O componente agora está disponível para ser utilizado pelas aplicações.

9 CRIANDO O COMPONENTE *TIMAGE3D*

O Componente *TImage3D* foi derivado do componente *TImage* e, em seguida, modificado por meio da inserção de novas propriedades e métodos que permitem o desenho em 3D.

Os métodos *MoveTo* e *LineTo*, por exemplo, herdados do *TCanvas*, ganharam seus equivalentes (*Mover* e *Ligar*) para o desenho em 3D, nesse novo componente. A projeção de um ponto no espaço é feita pelo método *Converte*, que utiliza as matrizes de rotação definidas anteriormente neste trabalho.

9.1 Métodos e propriedades do TImage3D

Os métodos criados para a classe *TImage3D* são:

- *TImage3D* – é o construtor da classe, em que podem ser definidos, dentre outras coisas, os valores iniciais das propriedades;
- *Mover* – move a caneta para a projeção de um ponto no espaço (*x,y,z*), de acordo com os ângulos de rotação;
- *Ligar* – move a caneta para a projeção de um ponto no espaço (*x,y,z*), de acordo com os ângulos de rotação. Desenha uma linha reta entre a antiga posição da caneta e a nova;
- *Cor* – ajusta a cor da caneta;
- *Limpa* – apaga o conteúdo do Canvas e pinta o fundo de amarelo;
- *DesenhaEixos* – desenha a projeção dos eixos *X*, *Y* e *Z* de acordo com os ângulos de rotação;
- *DesenhaFuncao* – desenha a projeção da função especificada pelo método *SetFuncao*, utilizando os limites impostos pelas propriedades *XInicial*, *XFinal*, *YInicial* e *YFinal*. O número de divisões é especificado pela propriedade *NumeroDeDivisoes*. A projeção é definida pelos ângulos de rotação;
- *Desenha* – limpa o Canvas, desenha os eixos e desenha a função;
- *Converte* – converte as coordenadas de um ponto no espaço (*x,y,z*) para um ponto no plano (*x,y*), de acordo com os ângulos de rotação;
- *SetFuncao* – ajusta o ponteiro para a função que será plotada.

As propriedades acrescentadas à classe *TImage3D* são:

- *RotacaoX* – rotação em torno do eixo *X*;
- *RotacaoY* – rotação em torno do eixo *Y*;

- *RotacaoZ* – rotação em torno do eixo *Z*;
- *Zoom* – aproximação da imagem;
- *DeslocX* – deslocamento na imagem na direção *X*;
- *DeslocY* – deslocamento na imagem na direção *Y*;
- *NumeroDeDivisoes* – número de divisões da malha (quadriculado) utilizada para o traçado da função;
- *XInicial* – valor inicial do intervalo de *X*;
- *XFinal* – valor final do intervalo de *X*;
- *YInicial* – valor inicial do intervalo de *Y*;
- *YFinal* – valor final do intervalo de *Y*.

Além da classe *TImage3D*, que define o novo componente, foram criadas também duas classes auxiliares (*Ponto2D* e *Ponto3D*) para armazenar as coordenadas de um ponto no plano (2D) e no espaço (3D), respectivamente.

9.2 Traçado da função

Para o traçado de uma função em 3D, é necessário delimitar as faixas de *x* e *y* que serão utilizadas. Em seguida, é necessário calcular, para cada par *x*, *y*, a coordenada *z* correspondente à função que se deseja traçar.

Uma maneira fácil de elaborar o traçado é fixar cada valor de *x* e variar os valores de *y*, interligando-se os pontos (*x*,*y*,*z*) encontrados por segmentos de reta. Dessa forma, obtém-se algo parecido com a primeira imagem da Figura 8.

Em seguida faz-se o inverso, fixando-se cada valor de *y* e variando-se os valores de *x*, obtendo algo como a segunda imagem da Figura 8.

O resultado da união dessas duas imagens nos fornece o gráfico da função em 3D.

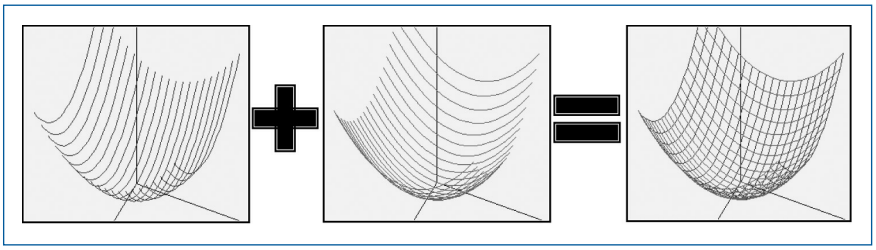

Figura 8 Traçado da função

9.3 O código completo do componente *TImage3D*

QUADRO 5

Listagem final do arquivo de cabeçalho *Image3D*.h.

```
#ifndef Image3DH
#define Image3DH
#include <sysutils.hpp>
#include <controls.hpp>
#include <classes.hpp>
#include <forms.hpp>
#include <StdCtrls.hpp>

class TPonto3D
{
 public:
  TPonto3D(); // Construtor
  void SetX(double x);
  void SetY(double y);
  void SetZ(double z);
  void SetXYZ(double x, double y, double z);
  double GetX();
  double GetY();
  double GetZ();
 private:
  double x;
  double y;
  double z;
};

class TPonto2D
{
 public:
  TPonto2D(); // Construtor
  void SetX(double x);
  void SetY(double y);
  double GetX();
  double GetY();
 private:
  double x;
  double y;
};
```

```
class PACKAGE TImage3D : public TImage
{
private:
  int FWx;       // Rotação em X
  int FWy;       // Rotação em Y
  int FWz;       // Rotação em Z
  int FDx;       // Deslocamento em X
  int FDy;       // Deslocamento em Y
  int FZoom;     // Aproximação
  int FNumDiv;   // Número de divisões
  double FXi;    // X inicial
  double FXf;    // X final
  double FYi;    // Y inicial
  double FYf;    // Y final
  double sx, sy, sz, cx, cy, cz, aa;
  void SinCos();
  TPonto2D __fastcall Converte(TPonto3D P3);
  double (*Ffuncao)(double x, double y);
protected:
public:
  // Construtor
  virtual __fastcall TImage3D(TComponent* Owner);
  // Métodos para Leitura
  int __fastcall GetWX();
  int __fastcall GetWY();
  int __fastcall GetWZ();
  // Métodos para Gravação
  void __fastcall SetWX(int wx);
  void __fastcall SetWY(int wy);
  void __fastcall SetWZ(int wz);
  // Métodos para Desenho
  void __fastcall Mover(double x, double y, double z);
  void __fastcall Ligar(double x, double y, double z);
  void __fastcall Cor(TColor c);
  void __fastcall Limpa();
  void __fastcall DesenhaEixos();
  void __fastcall DesenhaFuncao();
  void __fastcall Desenha();
  // Definição da Função Matemática
  void __fastcall SetFuncao(double (*f)(double x, double y));
__published:
  // Propriedades
  __property int RotacaoX = {read=GetWX, write=SetWX,
default=360};
  __property int RotacaoY = {read=GetWY, write=SetWY,
default=360};
```

```
  __property int RotacaoZ = {read=GetWZ, write=SetWZ,
default=360};
  __property int Zoom = {read=FZoom, write=FZoom};
  __property int NumeroDeDivisoes = {read=FNumDiv, write=FNumDiv};
  __property int DeslocX = {read=FDx, write=FDx};
  __property int DeslocY = {read=FDy, write=FDy};
  __property double XInicial = {read=FXi, write=FXi};
  __property double XFinal = {read=FXf, write=FXf};
  __property double YInicial = {read=FYi, write=FYi};
  __property double YFinal = {read=FYf, write=FYf};
};

#endif
```

QUADRO 6

Listagem final do arquivo fonte *Image3D*.cpp

```
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Image3D.h"
#pragma package(smart_init);

static inline void ValidCtrCheck(TImage3D *)
{
 new TImage3D(NULL);
}

double FuncaoDefault(double x, double y)
{
 return (x*x + y*y)/5.0;
}

//--------------------------
// Métodos da Classe TPonto3D
//--------------------------

TPonto3D::TPonto3D() // Construtor
{
 x = 0; y = 0; z = 0;
}
void TPonto3D::SetX(double x1)
{
 x = x1;
```

```
}
void TPonto3D::SetY(double y1)
{
 y = y1;
}
void TPonto3D::SetZ(double z1)
{
 z = z1;
}
void TPonto3D::SetXYZ(double x1, double y1, double z1)
{
 x = x1; y = y1; z = z1;
}
double TPonto3D::GetX()
{
 return x;
}
double TPonto3D::GetY()
{
 return y;
}
double TPonto3D::GetZ()
{
 return z;
}

//-------------------------
// Métodos da Classe TPonto2D
//-------------------------

TPonto2D::TPonto2D() // Construtor
{
 x = 0; y = 0;
}
void TPonto2D::SetX(double x1)
{
 x = x1;
}
void TPonto2D::SetY(double y1)
{
 y = y1;
}
double TPonto2D::GetX()
{
 return x;
}
```

```
double TPonto2D::GetY()
{
 return y;
}

//---------------------------
// Métodos do Componente TImage3D
//---------------------------

__fastcall TImage3D::TImage3D(TComponent* Owner)
 : TImage(Owner)
// Construtor
{
 FWx = 100;
 FWy = 180;
 FWz = 75;
 Width = 300;
 Height = 300;
 Zoom = 10;
 NumeroDeDivisoes = 20;
 DeslocX = 0;
 DeslocY = 0;
 XInicial = 0.0;
 XFinal = 0.0;
 YInicial = 0.0;
 YFinal = 0.0;
 SetFuncao(FuncaoDefault);
}

void TImage3D::SinCos()
{
 cx = cos(FWx * M_PI/180);
 cy = cos(FWy * M_PI/180);
 cz = cos(FWz * M_PI/180);
 sx = sin(FWx * M_PI/180);
 sy = sin(FWy * M_PI/180);
 sz = sin(FWz * M_PI/180);
 aa = sx * sy;
}

int __fastcall TImage3D::GetWX()
{
 return FWx;
}
int __fastcall TImage3D::GetWY()
{
```

```
 return FWy;
}
int __fastcall TImage3D::GetWZ()
{
 return FWz;
}

void __fastcall TImage3D::SetWX(int wx)
{
 FWx = wx;
 SinCos();
}
void __fastcall TImage3D::SetWY(int wy)
{
 FWy = wy;
 SinCos();
}
void __fastcall TImage3D::SetWZ(int wz)
{
 FWz = wz;
 SinCos();
}

TPonto2D __fastcall TImage3D::Converte(TPonto3D P3)
{
 TPonto2D P2;
 P3.SetX(P3.GetX() * FZoom);
 P3.SetY(P3.GetY() * FZoom);
 P3.SetZ(P3.GetZ() * FZoom);
 P2.SetX((cy * cz) * P3.GetX()
         - (cy * sz) * P3.GetY()
         + sy * P3.GetZ() + FDx);
 P2.SetY((aa * cz + cx * sz) * P3.GetX()
         + (-aa * sz + cx * cz) * P3.GetY()
         - (sx * cy) * P3.GetZ() + FDy);
 P2.SetX(P2.GetX() + Width / 2); // Centraliza em X
 P2.SetY(Height / 2 - P2.GetY()); // Centraliza em Y
 return P2;
}

void __fastcall TImage3D::Mover(double x, double y, double z)
{
 TPonto2D P2;
 TPonto3D P3;
 P3.SetXYZ(x, y, z);
 P2 = Converte(P3);
```

```
 Canvas->MoveTo(P2.GetX(), P2.GetY());
}

void __fastcall TImage3D::Ligar(double x, double y, double z)
{
 TPonto2D P2;
 TPonto3D P3;
 P3.SetXYZ(x, y, z);
 P2 = Converte(P3);
 Canvas->LineTo(P2.GetX(), P2.GetY());
}

void __fastcall TImage3D::Cor(TColor c)
{
 Canvas->Pen->Color = c;
}

void __fastcall TImage3D::Limpa()
{
 Canvas->Brush->Color = (TColor)0x00C0FFFF;
 Canvas->FillRect(Rect(0, 0, Width, Height));
}

void __fastcall TImage3D::DesenhaEixos()
{
 Cor(clBlue);  // Eixo X
 Mover(0, 0, 0);
 Ligar(500, 0, 0);
 Cor(clRed);  // Eixo Y
 Mover(0, 0, 0);
 Ligar(0, 500, 0);
 Cor(clGreen); // Eixo Z
 Mover(0, 0, 0);
 Ligar(0, 0, 500);
}

void __fastcall TImage3D::DesenhaFuncao()
{
 double X, Y, Z, PX, PY;
 if (FNumDiv==0)
  return;
 PX = (FXf - FXi) / FNumDiv;
 PY = (FYf - FYi)/ FNumDiv;
 // Traça curvas azuis
 Cor(clBlue);
 X = FXi;
```

```
 do{
   Y = FYi;
   do{
     Z = Ffuncao(X, Y);
     if (Y == FYi) Mover(X, Y, Z);
     else Ligar (X, Y, Z);
     Y = Y + PY;
   } while (Y <= FYf && PY>0);
   X = X + PX;
 } while (X <= FXf && PX>0);
 // Traça curvas vermelhas
 Cor(clRed);
 Y = FYi;
 do{
   X = FXi;
   do{
     Z = Ffuncao(X, Y);
     if (X == FXi) Mover(X, Y, Z);
     else Ligar (X, Y, Z);
     X = X + PX;
   } while (X <= FXf && PX>0);
   Y = Y + PY;
 } while (Y <= FYf && PY>0);
}

void __fastcall TImage3D::Desenha()
{
 Limpa();
 DesenhaEixos();
 DesenhaFuncao();
}

void __fastcall TImage3D::SetFuncao(double (*f)(double x, double
y))
{
 Ffuncao = f;
}

//---------------------
// Registro do Componente
//---------------------

namespace Image3d
{
 void __fastcall PACKAGE Register()
 {
```

```
   TComponentClass classes[1] = {__classid(TImage3D)};
   RegisterComponents("Samples", classes, 0);
 }
}
```

9.4 Um exemplo de aplicação do *TImage3D*

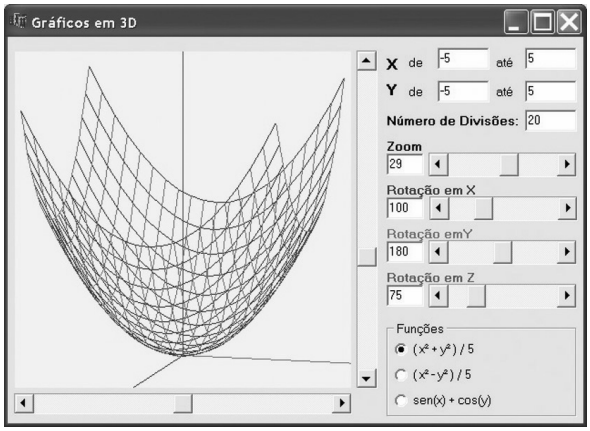

Figura 9 Tela do programa de exemplo

QUADRO 7

Listagem do arquivo exemplo.cpp

```
#include <vcl.h>
#include <math.h>
#pragma hdrstop
#include "Exemplo.h"
#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
```

```
//---------------------------
// Definição das funções matemáticas
//---------------------------

double f1(double x, double y)
{
 return (x*x+y*y)/5.0;
}

double f2(double x, double y)
{
 return (x*x-y*y)/5.0;
}

double f3(double x, double y)
{
 return (sin(x)+cos(y));
}

//-----------------------
// Implementação dos Eventos
//-----------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
 sbDX->Max = Image3D1->Width;
 sbDX->Min = -Image3D1->Width;
 sbDX->Position = 0;
 sbDY->Max = Image3D1->Height;
 sbDY->Min = -Image3D1->Height;
 sbDY->Position = 0;
 Image3D1->SetFuncao(f1);
 Image3D1->Desenha();
}

void __fastcall TForm1::sbRotXChange(TObject *Sender)
{
 edtRotX->Text = IntToStr(sbRotX->Position);
 Image3D1->SetWX(sbRotX->Position);
 Image3D1->Desenha();
}

void __fastcall TForm1::sbRotYChange(TObject *Sender)
{
 edtRotY->Text = IntToStr(sbRotY->Position);
 Image3D1->SetWY(sbRotY->Position);
```

```
 Image3D1->Desenha();
}

void __fastcall TForm1::sbRotZChange(TObject *Sender)
{
 edtRotZ->Text = IntToStr(sbRotZ->Position);
 Image3D1->SetWZ(sbRotZ->Position);
 Image3D1->Desenha();
}

void __fastcall TForm1::sbDXChange(TObject *Sender)
{
 Image3D1->DeslocX = sbDX->Position;
 Image3D1->Desenha();
}

void __fastcall TForm1::sbDYChange(TObject *Sender)
{
 Image3D1->DeslocY = (-1)*sbDY->Position;
 Image3D1->Desenha();
}

void __fastcall TForm1::sbZoomChange(TObject *Sender)
{
 edtZoom->Text = IntToStr(sbZoom->Position);
 Image3D1->Zoom = sbZoom->Position;
 Image3D1->Desenha();
}

void __fastcall TForm1::edtXiChange(TObject *Sender)
{
 double aux;
 if (TryStrToFloat(edtXi->Text,aux)){
    Image3D1->XInicial = aux;
    Image3D1->Desenha();
 }
}

void __fastcall TForm1::edtXfChange(TObject *Sender)
{
 double aux;
 if (TryStrToFloat(edtXf->Text,aux)){
    Image3D1->XFinal = aux;
    Image3D1->Desenha();
 }
}
```

```
void __fastcall TForm1::edtYiChange(TObject *Sender)
{
 double aux;
 if (TryStrToFloat(edtYi->Text,aux)){
    Image3D1->YInicial = aux;
    Image3D1->Desenha();
 }
}

void __fastcall TForm1::edtYfChange(TObject *Sender)
{
 double aux;
 if (TryStrToFloat(edtYf->Text,aux)){
    Image3D1->YFinal = aux;
    Image3D1->Desenha();
 }
}

void __fastcall TForm1::edtDivChange(TObject *Sender)
{
 int aux;
 if (TryStrToInt(edtDiv->Text,aux)){
   if (aux<=0) aux=1;
   Image3D1->NumeroDeDivisoes = aux;
   Image3D1->Desenha();
 }
}

void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
 switch (RadioGroup1->ItemIndex){
 case 0:
  Image3D1->SetFuncao(f1);
  break;
 case 1:
  Image3D1->SetFuncao(f2);
  break;
 case 2:
  Image3D1->SetFuncao(f3);
  break;
 }
 Image3D1->Desenha();
}
```

Ajuste o conteúdo inicial das *Edits* e a posição das *ScrollBars* de acordo com o valor de cada propriedade do *Image3D* no *Object Inspector*.

10 CONSIDERAÇÕES FINAIS

Os códigos deste trabalho foram escritos utilizando-se o C++ *Builder* 6 versão *Personal*. Outras versões podem apresentar pequenas diferenças na manipulação do ambiente ou na implementação dos códigos.

Considere a criação e reutilização de um componente derivado, por exemplo, do *TImage* ou do *TImage3D* de forma a traçar gráficos de curvas ou caminhos no espaço euclidiano R^3 e também outros elementos gráficos do espaço tridimensional.