Rectilinear Motion

Homework2

Miguel Tlapa Juárez

31/01/2014

This document describes the system architecture and design about the body controller module, it's have block diagram and flowchart to describe software and hardware architecture.

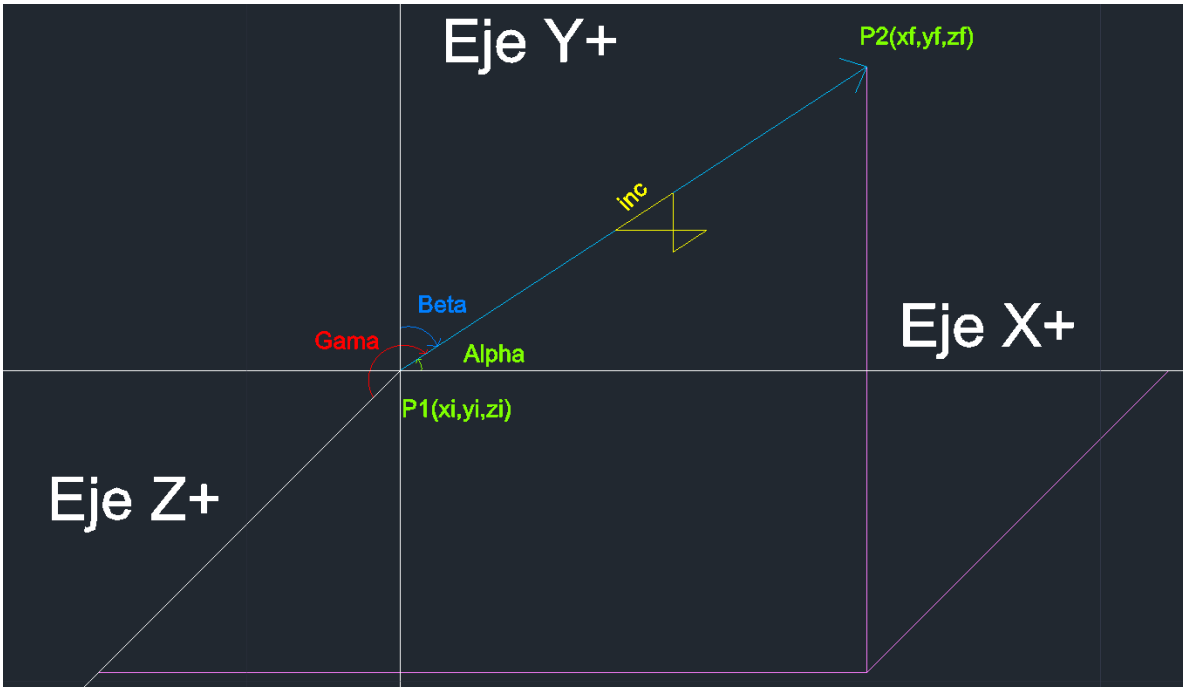
Revision History

Date	Revision Number	Author/Editor	Modifications
January 2014	0.1	Miguel Tlapa	Created file

Disclaimers

1. Explanation

1.1 RectilinearMotion3d.cpp



inc x= Spacial Physics Increment

delay = Drawing Time

Data:

tx = 10 [seg]

xi = -5.0 = Initial Position

xf = 5.0 = Final Position

x = xi = Change of variable

//

yi = -5.0 = Initial Position

yf = 5.0 = Final Position

y = yi = Change of variable

//////////////////////////////////////

zi = -5.0 = Initial Position

zf = -10.0 = Final Position

z = zi = Change of variable

Equations

First we need to calculate the distance between 2 points

P2 and P1

d = (sqrt (pow(xf - xi, 2) + pow(yf - yi, 2) + pow(zf - zi, 2)))

Note pow(xf-xi) = pow(xi-xf)

Increment x is definede trough velocity of movement.

inc = small increment whith respect theta angle in the plane xy

inc = d/t = 10units/10seg = 1/seg

Using Direction Cosines

Cos alpha = x/d = xf-xi/d

Cos alpha = y/d = yf-yi/d

Cos gama = z/d = zf-zi/d

Angles are:

Alpha = acos(xf-xi/d)

Beta = acos(yf-xi/d)

Gama = acos(zf-zi/d)

The increments in xyz are:

inc x = inc * cos alpha

inc y = inc * cos beta

inc z = inc * cos gama

Rule of tree

distance ---> time

incx ----> [tseg]

[tseg] = Time of Refresh that use GPU to draw objects.

[tseg] = incx * time/ distance However we need to multiply x 1000 because the time should be in milisecods

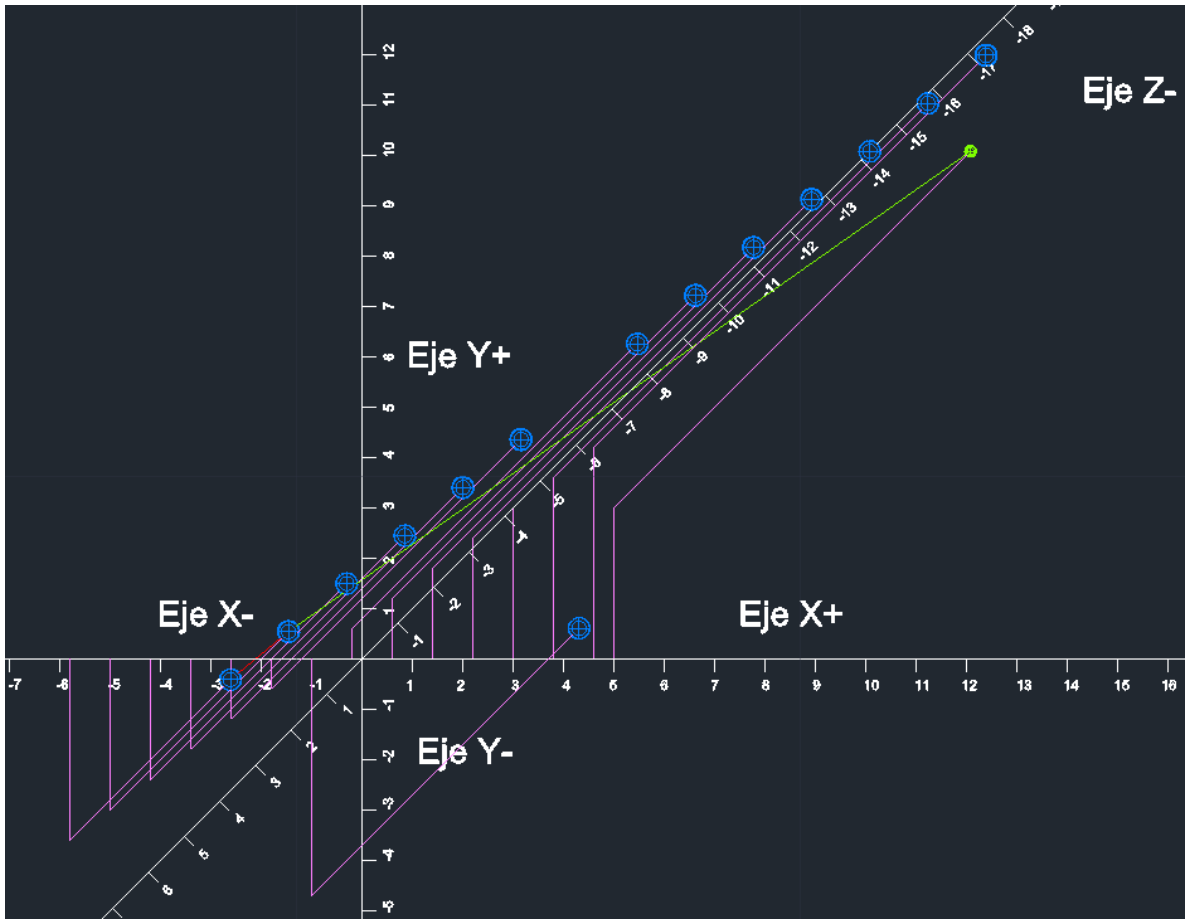
[tseg] = incx * time/ distance * 1000

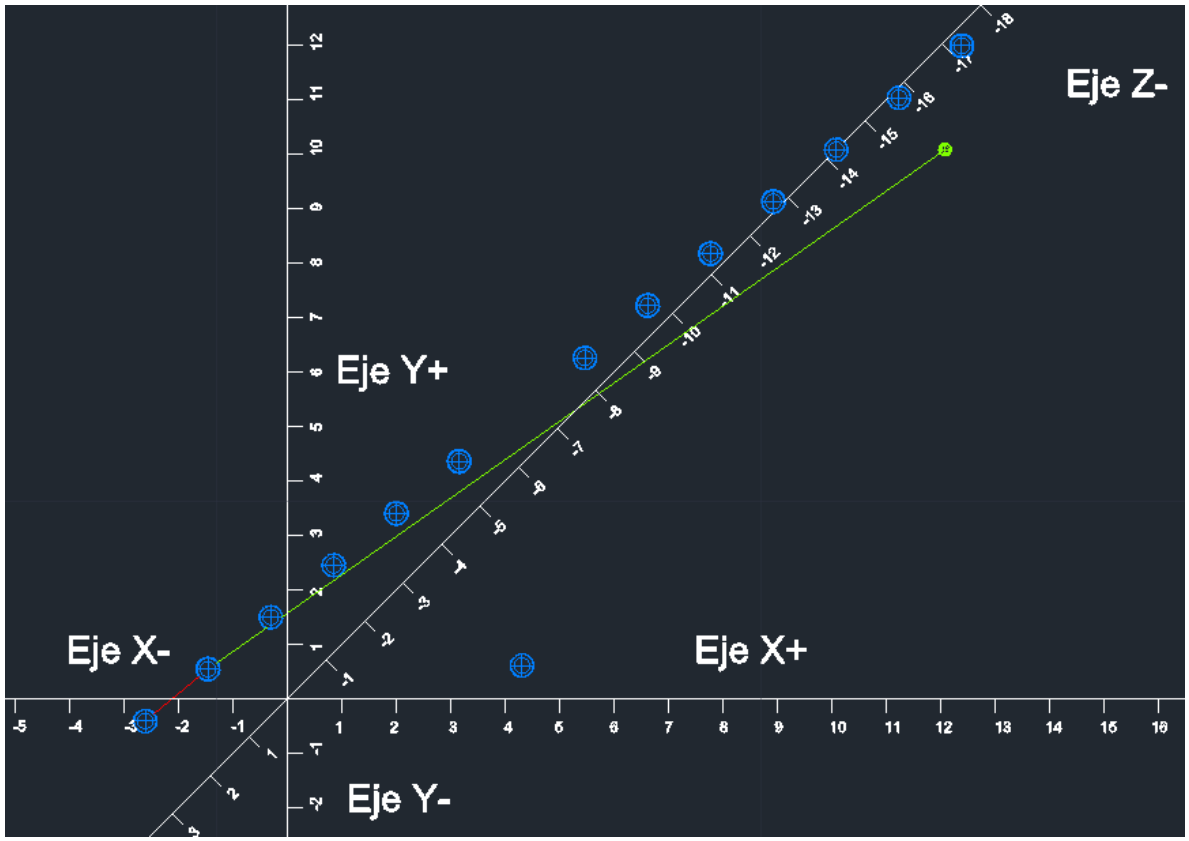
With this relation we can adjust the Time of Refresh with Increment X.

I captured the data in each iteration

```
x -IncX -5.8y -IncY -3.6 z -IncZ -4.5
x -IncX -5y -IncY -3 z -IncZ -5
x -IncX -4.2y -IncY -2.4 z -IncZ -5.5
x -IncX -3.4y -IncY -1.8 z -IncZ -6
x -IncX -2.6y -IncY -1.2 z -IncZ -6.5
x -IncX -1.8y -IncY -0.6 z -IncZ -7
However this value is out of range
x -IncX -1y -IncY -4.76837e-07 z -IncZ -7.5

x -IncX -0.2y -IncY 0.599999 z -IncZ -8
x -IncX 0.6y -IncY 1.2 z -IncZ -8.5
x -IncX 1.4y -IncY 1.8 z -IncZ -9
x -IncX 2.2y -IncY 2.4 z -IncZ -9.5
x -IncX 3y -IncY 3 z -IncZ -10
x -IncX 3.8y -IncY 3.6 z -IncZ -10.5
x -IncX 4.6y -IncY 4.2 z -IncZ -11
x -IncX 4.6y -IncY 4.2 z -IncZ -11
x -IncX 4.6y -IncY 4.2 z -IncZ -11
```


Eje Z-
Eje Y+
Eje X-
Eje X+
Eje Y-


Eje Z-
Eje Y+
Eje X-
Eje X+
Eje Y-

```
// ************************************************************
// ** Instituto Tecnológico y de Estudios Superiores de Occidente **
// **        ITESO, Universidad Jesuita de Guadalajara          **
// **                                                           **
// **     Especialidad en Computación Gráfica para Videojuegos    **
// **    ECG2227A - Física para el Modelado de Sistemas Reales    **
// **                                                           **
// **                 Homework Number2                      **
// **                 MIGUEL TLAPA JUAREZ                     **
// **                 RECTILINEAR MOTION3D                     **
// ************************************************************

#include <math.h>
#include <iostream>
#include "PhAPI.h"
using namespace PhAPI;
using namespace std;

// ** Definición de Variables para Manipulación del Entorno
int width = 1000;                                   // Ancho de la Ventana
int height = 800;                                   // Alto de la Ventana

float xi = -5.0;                                    // Posición Inicial en el Eje
X
float xf = 5.0;                                          // Posición Final en el
Eje X
```

float x = xi; // Variable de Intercambio en X

float yi = -3.0; // Posición Inicial en el Eje Y

float yf = 3.0; // Posición Final en el Eje Y

float y = yi; // Variable de Intercambio en Y

float zi = -5.0; // Posición Inicial en el Eje Z

float zf = -10.0; // Posición Final en el Eje Z

float z = zi; // Variable de Intercambio en Z

int t = 10; // Tiempo de Trayectoria [segundos]

float d = (sqrt (pow(xf - xi, 2) + pow(yf - yi, 2) + pow(zf - zi, 2))); // Distancia de Trayectoria [segmentos]

float inc = (float) d / t; // Incremento en la Trayectoria [segmentos/segundo]

float tseg = (float) inc * t / d; // Tiempo para un solo segmento [segundos]

int delay = int (tseg * 1000); // Tiempo de Retraso [milisegundos]

```
float alfa  = acos ((xf-yi)/(d));     // Ángulo de Inclinación [radianes]

float beta  = acos ((yf-yi)/(d));    // Ángulo de Inclinación [radianes]
float gama  = acos ((zf-zi)/(d));  // Ángulo de Inclinación [radianes]

float incX = inc * cos(alfa);                    // Incremento en el Eje X [segmentos]
float incY = inc * cos(beta);                    // Incremento en el Eje Y [segmentos]
float incZ = inc * cos(gama);                    // Incremento en el Eje Y [segmentos]

// ** Dibujar las Primitivas
void DibujarVentana()
{
        // Dibuja líneas de Referencia
        setColor(1.0, 0.0, 1.0);
        drawLine(xi, yi, zi, xf, yf, zf);

        // Dibuja la Primitiva en forma de Esfera
        setColor(0.0, 0.0, 1.0);
//      cout << "Alpha Angle "  << alfa << endl;
//      cout << "Beta Angle "  << beta << endl;
```

```
//        cout << "Gama Angle "  << gama << endl;
//        cout << "IncX "  << incX << endl;
//        cout << "IncY "  << incY << endl;
//        cout << "IncZ "  << incZ << endl;
        cout << "x -IncX "  << x -incX << "y -IncY " << y -incY <<" z -IncZ " << z -
incZ << endl;

        drawSphere(x - incX, y - incY, z - incZ, 0.2);

        // Modifica la Posición de la Primitiva
        if ( x <= xf )
        {
                x += incX;
           y += incY;
           z += incZ;
        }
}

// ** Programa Principal
int main()
{
        setDelay(delay);
        createWindow("Practica en 3 Ejes", width, height);
        setBackground(0.7, 0.7, 0.7);
```

```
        setDisplay(DibujarVentana);
        showWindow();
        return 0;
}
```