Universidad Nacional de San Antonio Abad del Cusco
Departamento Académico de Informática
COMPUTACIÓN GRAFICA I
Práctica Nº 01

REPRESENTACIÓN EXPLÍCITA Y PARAMÉTRICA DE LA CURVA

Iván C. Medrano Valencia

1. OBJETIVO.

- Conocer cómo trazar curvas explícitas y paramétricas.

1.1. Representación explícita de la curva.

Para representar **líneas** y **curvas** matemáticamente se utiliza un polinomio que puede tomar la forma:

$$y = f(x),$$

donde **f (x)** consta de potencias de **x** solamente. Un polinomio **lineal** (es decir, de **primer grado**) se utiliza para representar una **línea** como:

$$y = mx + b,$$

donde **m** es la pendiente y **b** es la intersección con el eje **y**. Un polinomio **cuadrático** (es decir, de **segundo grado**) se utiliza para representar una **curva** como:

$$y = ax^2 + bx + c,$$

Además, también, se utiliza un polinomio **cúbico** (es decir, de **tercer grado**) para representar una **curva** como:

$$y = ax^3 + bx^2 + cx + d,$$

1.2. Representación paramétrica.

Para curvas 2D cuadráticas, se utilizan los siguientes polinomios cuadráticos para obtener un punto de la curva $\dot{p}(t) = [x(t), y(t)]^T$:

$$x(t) = a_x t^2 + b_x t + c_x,$$
$$y(t) = a_y t^2 + b_y t + c_y,$$

donde a_x, b_x, c_x, a_y, b_y y c_y son coeficientes para las direcciones x e y respectivamente; y $t \in [0, 1]$ es un parámetro que determina la ubicación del punto de la curva $\dot{p}(t) = [x(t), y(t)]_T$ a lo largo de la curva. En forma vectorial, se puede reescribir como:

$$x(t) = \underbrace{\begin{bmatrix} t^2 & t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix}}_{\mathbf{x}} = \mathbf{t} \cdot \mathbf{x}^T,$$

$$y(t) = \underbrace{\begin{bmatrix} t^2 & t & 1 \end{bmatrix}}_{\mathbf{t}} \underbrace{\begin{bmatrix} a_y \\ b_y \\ c_y \end{bmatrix}}_{\mathbf{y}} = \mathbf{t} \cdot \mathbf{y}^T,$$

2. DESARROLLO DE LA PRÁCTICA.

2.1. Curvas explícitas.

Trazar las siguientes curvas:

a) $y = x^2 + x + 1$

b) $y = 4.1579x^2 - 7.2158x$

c) $y = 8\sin(x)$

d) $y = 4\cos(3x)$

e) $y = x^2 - x - 2$

f) $y = x^2 - 2x - 3$

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define WIDTH 500    /* Ancho ventanas en pixels */
#define HEIGHT 500   /* Alto ventanas en pixels */


#define MAX_CPTS  100

GLfloat cpts[MAX_CPTS][2];
int ncpts = MAX_CPTS;

void generaPuntos()
{
	float x = 0.0;
	for (int i = 0; i < ncpts; i++)
	{

		cpts[i][0] = x;
		cpts[i][1] = x * x + x + 1; //--función de la curva
		x = x + 0.2;
	}
}
/* Dibuja los ejes de referencia */
void ejes(void) {
	glBegin(GL_LINES);
	glVertex2i(0, -10); //--eje Y
	glVertex2i(0, 80);

	glVertex2i(-5, 0); //--eje X
	glVertex2i(35, 0);
}

/* Render de la ventana */
void display(void) {

	int i;
	//glClear(GL_COLOR_BUFFER_BIT);
```

```
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1.0, 1.0, 0.0); //--color amarillo
        ejes();
        glColor3f(1.0, 0.0, 0.0);
        glPointSize(2.0);
        generaPuntos();
        glBegin(GL_LINES);
        for (i = 0; i < ncpts-1; i++)
        {
                glVertex2fv(cpts[i]);
                glVertex2fv(cpts[i + 1]);
        }
        glEnd();

        glutSwapBuffers();

}
void ini(void) {
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        gluOrtho2D(-50.0, 200.0, -200.0, 499.0);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glScalef(5.0, 5.0,0.0);

}
int main(int argc, char** argv) {

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(WIDTH, HEIGHT);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Curvas explícitas 2D");
        glutDisplayFunc(display);
        ini();

        glutMainLoop();
        return 0;
}
```

2.2. Curva Explícita a obteniendo su ecuación por 3 puntos.

Considere una curva cuadrática cuyos puntos finales son [0, 0]T y [2, 2.2]T. Si el punto [1.9, 1.3]T está en la curva, obtenga su ecuación polinomial y dibuje la curva

```
#include <GL/glut.h>
#include <stdio.h>
#include <math.h>
#define WIDTH 500    /* Ancho ventanas en pixels */
#define HEIGHT 500   /* Alto ventanas en pixels */

#define MAX_CPTS  11
//

GLfloat X0 = 0.0;
GLfloat Y0 = 0.0;
GLfloat X1 = 2.0;
GLfloat Y1 = 2.2;
GLfloat X2 = 1.9;
```

```
GLfloat Y2 = 1.3;

GLfloat cpts[MAX_CPTS][2];
int ncpts = MAX_CPTS;

GLfloat a()
{
	return (X0 * (Y1 - Y2) + Y0 * (X2 - X1) + X1 * Y2 - X2 * Y1) / ((X1
- X0) * (X2 - X0) * (X2 - X1));
}
GLfloat b()
{
	return ((Y2 - Y0) / (X2 - X0)) - (X2 + X0) * a();
}
GLfloat c()
{
	return Y0 - (a() * (X0 * X0)) - b() * X0;
}
void generaPuntos()
{
	printf("a=%f, b=%f, c=%f ", a(), b(), c());
	GLfloat X = 0.0;
	for (int i = 0; i < ncpts; i++)
	{

		cpts[i][0] = X;
		cpts[i][1] = a() * (X * X) + b() * X + c(); //--función de la
curva
		X = X + 0.2;
	}
}
/* Dibuja los ejes de referencia */
void ejes(void) {
	glBegin(GL_LINES);
	glVertex2i(0, -10); //--eje Y
	glVertex2i(0, 80);

	glVertex2i(-5, 0); //--eje X
	glVertex2i(35, 0);
}

/* Render de la ventana */
void display(void) {

	int i;
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glColor3f(1.0, 1.0, 0.0); //--color amarillo
	ejes(); //--dibuja los ejes
	glColor3f(1.0, 0.0, 0.0);
	glPointSize(2.0);
	////--obtener ptos de la matriz
	//X0 = ptos[0][0];
	//Y0 = ptos[0][1];
	//X1 = ptos[1][0];
	//Y1 = ptos[1][1];
	//X2 = ptos[2][0];
	//Y2 = ptos[2][1];

	generaPuntos(); //--genera los ptos a plotear
	//--dibuja una linea entre cada par de ptos.
	glBegin(GL_LINES);
	for (i = 0; i < ncpts - 1; i++)
```

```
        {
                glVertex2fv(cpts[i]);
                glVertex2fv(cpts[i + 1]);
        }
        glEnd();
        //--dibuja los puntos de control
        glColor3f(0.0, 1.0, 0.0);
        glPointSize(5.0);
        glBegin(GL_POINTS);
        {
                glVertex2f(X0, Y0);
                glVertex2f(X1, Y1);
                glVertex2f(X2, Y2);
        }
        glEnd();
        glutSwapBuffers();

}
void ini(void) {
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        gluOrtho2D(-5.0, 5.0, -5.0, 10.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //glScalef(5.0, 5.0, 0.0);
}
int main(int argc, char** argv) {

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(WIDTH, HEIGHT);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Curva explícita 2D a partir de 3 puntos");
        glutDisplayFunc(display);
        ini();

        glutMainLoop();
        return 0;
}
```

2.3. Curva explícita con rotación.

Suponga que una curva explícita cuadrática pasa por los puntos $[2, 2]^T$, $[4, 5]^T$ y $[9, 8]^T$. Dibujar dicha curva y luego dibujar otra curva si los puntos se rotan 30°con respecto al origen.

```
#include <GL/glut.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <stdio.h>
#include <math.h>
#include <iostream>

#define WIDTH 500   /* Ancho ventanas en pixels */
#define HEIGHT 500  /* Alto ventanas en pixels */

#define MAX_CPTS  500

GLfloat X0 = 2.0;
```

```
GLfloat Y0 = 2.0;
GLfloat X1 = 4.0;
GLfloat Y1 = 5.0;
GLfloat X2 = 9.0;
GLfloat Y2 = 8.0;
GLfloat cpts[MAX_CPTS][2];
int ncpts = MAX_CPTS;
GLint nroPtos;

GLfloat a()
{
  return (X0*(Y1-Y2)+Y0*(X2-X1)+X1*Y2-X2*Y1)/((X1-X0)*(X2-X0)*(X2 - X1));
}
GLfloat b()
{
  return ((Y2-Y0)/(X2-X0))-(X2+X0)*a();
}
GLfloat c()
{
  return Y0-(a()*(X0*X0))-b()*X0;
}
void generaPuntos(GLfloat Xini, GLfloat Xfin)
{
       GLfloat X = Xini;
       GLint i = 0;
       while (X <= Xfin)
       {
          cpts[i][0] = X;
          cpts[i][1] = a()*(X*X)+b()*X+--c(); //--función de la curva
          X = X + 0.05;
          i++;
       }

       nroPtos = i-1;
       printf("nro de ptos:, %d\n", nroPtos);
}
/* Dibuja los ejes de referencia */
void ejes(void) {
       glBegin(GL_LINES);
       glVertex2i(0, -10); //--eje Y
       glVertex2i(0, 80);

       glVertex2i(-5, 0); //--eje X
       glVertex2i(35, 0);
       glEnd();
}

void dibujaCurvaRotada()
{
   int i;
   //---definición de los puntos a rotar
   glm::vec4 pto0(X0, Y0, 0.0f, 1.0f);
   glm::vec4 pto1(X1, Y1, 0.0f, 1.0f);
   glm::vec4 pto2(X2, Y2, 0.0f, 1.0f);
   //--creamos la matriz identidad
   glm::mat4 matriz_ident = glm::mat4(1.0f);
   //--definimos la matriz de rotación
   glm::mat4 matriz_rotacion = glm::rotate(matriz_ident, glm::radians(30.0f),
 glm::vec3(0.0f, 0.0f, 1.0f));
   //--obtenemos los puntos rotados
   glm::vec4 pto0_rotado = matriz_rotacion * pto0;
   glm::vec4 pto1_rotado = matriz_rotacion * pto1;
```

```
    glm::vec4 pto2_rotado = matriz_rotacion * pto2;
    //--obtenemos coordenadas de los ptos rotados
    X0 = pto0_rotado.x;
    Y0 = pto0_rotado.y;
    X1 = pto1_rotado.x;
    Y1 = pto1_rotado.y;
    X2 = pto2_rotado.x;
    Y2 = pto2_rotado.y;
    //--mostrar puntos rotados
    printf("Puntos rotados:\n");
    printf("%f, %f\n ", X0, Y0);
    printf("%f, %f\n ", X1, Y1);
    printf("%f, %f\n ", X2, Y2);
    //--generar curva con puntos rotados
    generaPuntos(X0, X2);
    printf("Coeficiente del polinomio rotado:\n");
    printf("a=%f, b=%f, c=%f\n ", a(), b(), c());
    //--dibuja una linea entre cada par de ptos.
    glPointSize(1.5);
    glColor3f(0.0, 1.0, 1.0); //--color azul
    glBegin(GL_LINE_STRIP);
    for (i = 0; i <= nroPtos; i++)
    {
        glVertex2f(cpts[i][0], cpts[i][1]);
    }

    printf("%f %f\n", cpts[nroPtos][0], cpts[nroPtos][1]);
    glEnd();
    //--dibuja los puntos de control
    glColor3f(0.0, 0.0, 1.0);
    glPointSize(5.0);
    glBegin(GL_POINTS);
    {
        glVertex2f(X0, Y0);
        glVertex2f(X1, Y1);
        glVertex2f(X2, Y2);
    }
    glEnd();
}

void dibujaCurvaOriginal()
{
        int i;
        //--mostrar puntos originales
        printf("Puntos originales:\n");
        printf("%f, %f\n ", X0, Y0);
        printf("%f, %f\n ", X1, Y1);
        printf("%f, %f\n ", X2, Y2);
        //--coeficientes del polinomio
        printf("Coeficiente del polinomio original:\n");
        printf("a=%f, b=%f, c=%f\n ", a(), b(), c());
        generaPuntos(X0, X2); //--genera los ptos a plotear
        //--dibuja la curva.
        glPointSize(1.5);
        glColor3f(1.0, 0.0, 0.0); //--color rojo
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= nroPtos; i++)
        {
                glVertex2f(cpts[i][0], cpts[i][1]);
        }
        glEnd();
        //--dibuja los puntos de control
```

```
	glColor3f(0.0, 1.0, 0.0); //--color verde
	glPointSize(5.0);
	glBegin(GL_POINTS);
	{
		glVertex2f(X0, Y0);
		glVertex2f(X1, Y1);
		glVertex2f(X2, Y2);
	}
	glEnd();
}

/* Render de la ventana */
void display(void) {
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glColor3f(1.0, 1.0, 0.0); //--color amarillo
	ejes(); //--dibuja los ejes
	//--traza curva original
	dibujaCurvaOriginal();
	//--traza curva rotada
	dibujaCurvaRotada();
	glutSwapBuffers();

}
void ini(void) {
	glClearColor(0.0, 0.0, 0.0, 1.0);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	gluOrtho2D(-5.0, 30.0, -5.0, 30.0);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
	//glScalef(7.0, 7.0, 0.0);
}
int main(int argc, char** argv) {

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(WIDTH, HEIGHT);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("Curva explícita 2D a partir de 3 puntos");
	glutDisplayFunc(display);
	ini();
	glutMainLoop();
	return 0;
}
```

2.4. Curvas paramétricas.

Trazar la curva paramétrica que pasa por los puntos $[0, 0]^T$ y $[3, 3]^T$. Si el punto $[3, 8]^T$ está en la curva en t = 0.5.

```
#include <GL/glut.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
#include <stdio.h>
#include <math.h>
#include <iostream>

#define WIDTH 500   /* Ancho ventanas en pixels */
#define HEIGHT 500  /* Alto ventanas en pixels */
```

```
#define MAX_CPTS 100

GLfloat X0, X1, X2, Y0, Y1, Y2;
GLfloat cpts[MAX_CPTS][2];
GLint ncpts = MAX_CPTS;
GLint nroPtos;

//--puntos
glm::vec2 p0(0.0f, 0.0f);
glm::vec2 p1(3.0f, 8.0f);
glm::vec2 p2(3.0f, 3.0f);

//--coeficientes
glm::vec2 a(0.0, 0.0);
glm::vec2 b(0.0, 0.0);
glm::vec2 c(0.0, 0.0);

//--parámetro t
float t = 0.5;

void curvaParametrica()
{
	c = p0;
	a = (t * (p2 - p0) + (p0 - p1)) / (t * (1 - t));
	b = p2 - p0 - a;

	printf("a= %f, %f\n", a.x, a.y);
	printf("b= %f, %f\n", b.x, b.y);
	printf("c= %f, %f\n", c.x, c.y);
	//--generar los puntos de la curva
	GLfloat t = 0.0;
	GLint i = 0;
	while (t <= 1.0)
	{
	    //--función paramétrica
	    cpts[i][0] = a.x * (t * t) + b.x * t + c.x;
	    cpts[i][1] = a.y * (t * t) + b.y * t + c.y;
	    //--imprimir los puntos
	    printf("%f, %f\n ", cpts[i][0], cpts[i][1]);
	    t = t + 0.02;
	    i++;
	}
	nroPtos = i - 1;
	printf("nro de ptos:, %d\n", nroPtos);
	glPointSize(1.5);
	glColor3f(0.0, 1.0, 1.0);
	glBegin(GL_LINE_STRIP);
	for (i = 0; i <= nroPtos; i++)
	{
		glVertex2f(cpts[i][0], cpts[i][1]);
	}
	glEnd();

}

/* Dibuja los ejes de referencia */
void ejes(void) {
	glBegin(GL_LINES);
	glVertex2i(0, -100); //--eje Y
	glVertex2i(0, 150);
```

```
        glVertex2i(-10, 0); //--eje X
        glVertex2i(100, 0);
        glEnd();
}



/* Render de la ventana */
void display(void) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1.0, 1.0, 0.0); //--color amarillo
        ejes(); //--dibuja los ejes
        //--traza curva original
        curvaParametrica();
        glutSwapBuffers();

}
void ini(void) {
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        gluOrtho2D(-10, 100, -100.0, 150.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glScalef(7.0, 7.0, 0.0);
}
int main(int argc, char** argv) {

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(WIDTH, HEIGHT);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("Curva paramétrica cuadrática");
        glutDisplayFunc(display);
        ini();
        glutMainLoop();
        return 0;
}
```

3. TAREA

- Una curva cuadrática cuyos puntos finales son $[0, 0]^T$ y $[3, 3]^T$ se representa de forma paramétrica. Si el punto $[3, 8]^T$ está en la curva donde t = 0.7, muestre la curva generada.
- Si los puntos anteriores se rotan 30° con respecto al origen. Obtenga la nueva curva

4. BIBLIOGRAFIA

- ✓ Donald Hearn, M. Pauline Baker (2006); Gráficas por Computadora con OpenGL; Pearson Prentice Hall; 3° edición; Madrid.
- ✓ Rimon Elias (2019); Digital Media A problema-solving Aproach for Computer Graphics; eBook. New York
- ✓ Sumanta Guha (2019); Computer Graphics Through OpenGL; Third Edition; Taylor & Francis Group .
  NW.

✓ Gordon V.Scott, Clevenger Jhon (2019). Computer Graphics Programming in OpenGl with C++. Mercury Learning and Information.