# Universidad Nacional de San Antonio Abad del Cusco
# Departamento Académico de Informática
## COMPUTACIÓN GRAFICA I
# Práctica Nº 8

## Composicion de Transformaciones

1. **OBJETIVO.**
   - Conocer las instrucciones de OpenGL para realizar transformaciones.
   - Aplicar las instrucciones de transformación de OpenGL.

**1.1. Traslación.**

En OpenGL, para la traslación tanto en dos dimensiones(2D) como en tres dimensiones(3D) se utiliza la instrucción ***glTranslated***, en sus versiones de parámetros tipo float, ***glTranslatef***, como en su versión de parámetros tipo double, ***glTranslated***. Sus prototipos son los siguientes**:**

```
void glTranslatef(GL float x, GL float y, GL float z)
void glTranslated(GL double x, GL double y, GL double z)
```

***glTranslate*** mueve el sistema de coordenadas al punto especificado por x,y,z. La matriz activa es multiplicada por la matriz de traslación. Es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.

Por ejemplo, si queremos trasladar un triángulo a la posición (200,200) escribiríamos lo siguiente:

```
glTranslatef(200.0f,200.0f,0.0f);
glBegin(GL_TRIANGLES);
  glVertex3f(100.0f,100.0f,0.0f);
  glVertex3f(200.0f,100.0f,0.0f);
  glVertex3f(150.0f,150.0f,0.0f);
glEnd();
```

**1.2. Rotación.**

En OpenGL, la rotación se trabaja con la instrucción glRotate, en sus versiones de parámetros tipo float, ***glRotatef***, como en su versión de parámetros tipo double, ***glRotated***. Sus prototipos son los siguientes:

```
void glRotated(GLdouble angulo, GLdouble x, GLdouble y, GLdouble z)
void glRotatef(GLfloat angulo, GLfloat x, GLfloat y, GLfloat z)
```

***glRotate*** rota el sistema de coordenadas en el sentido de las manecillas del reloj en el ángulo especificado en la dirección del vector representado por las coordenadas x,y,z. La matriz activa es multiplicada por la matriz de rotación. Al igual que en el

caso de translación es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.
Supongamos que deseamos rotar un triángulo en 45° en torno al vector (0,0,1). Una vez rotado nuestro sistema para dibujar él triangulo debemos considerar la nueva ubicación del sistema de coordenadas del modelo, luego en base estas nuevas coordenadas realizamos nuestro dibujo. Nuestro sistema de coordenadas del modelo esta rotado 45 grados luego necesitamos calcular las coordenadas del triángulo para la posición deseada.

```
glRotatef(45,0,0,1);
glColor3f(0.5f, 1.0f, 0.7f);
glBegin(GL_TRIANGLES);
  glVertex2f(1.7f,0.0f);
  glVertex2f(3.7f,0.0f);
  glVertex2f(2.6f,1.0f);
glEnd();
```

**1.3. Escalamiento.**
En Opengl, el escalamiento se trabaja con la instrucción ***glScale***, en sus versiones de parámetros tipo float, ***glScalef***, como en su versión de parámetros tipo double, ***glScaled***. Sus prototipos son los siguientes:

***void glScaled(GLdouble x, GLdouble y, GLdouble z)***
***void glScalef(GLfloat x, GLfloat y, GLfloat z)***

***glScale*** escala el sistema de coordenadas en los factores indicados por x, y, z. La matriz activa es multiplicada por la matriz de escalamiento. Al igual que en el caso de traslación es posible utilizar glPushMatrix y glPopMatrix para guardar y recuperar el sistema de coordenadas.

Supongamos que deseamos escalar un triangulo en un factor 2 en todas las direcciones, entonces nuestro código sería el siguiente:

```
glScalef(2.0,2.0,2.0);
glColor3f(0.5f, 1.0f, 0.7f);
glBegin(GL_TRIANGLES);
  glVertex2f(-1.0f,0.0f);
  glVertex2f(1.0f,0.0f);
  glVertex2f(0.0f,1.0f);
glEnd();
```

**1.4. Composición de transformaciones.**
Es una secuencia de transformaciones, por ejemplo; si queremos trasladar y rotar un triangulo deberíamos hacer las siguientes instrucciones.

```
glRotatef(45, 0, 0, 1);
glTranslatef(200.0f, 200.0f, 0.0f);
glBegin(GL_TRIANGLES);
  glVertex2f(100.0f, 100.0f);
  glVertex2f(200.0f, 100.0f);
  glVertex2f(150.0f, 150.0f);
glEnd();
```

## 2. DESARROLLO DE LA PRÁCTICA.

### 2.1. Traslación .

```
#include<math.h>
#include<cstdlib>
#include<GL/glew.h>
#include<GL/glut.h>
#include<stdio.h>
using namespace std;

void Traslada()
{
      //--triangulo antes de trasladar
      glColor3f(0.5f, 1.0f, 0.7f);
      glBegin(GL_TRIANGLES);
      glVertex2f(100.0f, 100.0f);
      glVertex2f(200.0f, 100.0f);
      glVertex2f(150.0f, 150.0f);
      glEnd();
      //--triangulo después de trasladar
      glTranslatef(200.0f, 200.0f, 0.0f);
      glBegin(GL_TRIANGLES);
        glVertex3f(100.0f, 100.0f, 0.0f);
        glVertex3f(200.0f, 100.0f, 0.0f);
        glVertex3f(150.0f, 150.0f, 0.0f);
      glEnd();

}

//--despliega el gráfico
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);  //--establece el color de la ventana
      Traslada();
      glFlush(); //--fuerza la ejecución de los comandos de OpenGL
}
void myinit()
{
      glClearColor(1.0, 1.0, 1.0, 1.0);
      glColor3f(1.0, 0.0, 0.0);
      glPointSize(1.0); //--tamño de los puntos
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity(); //--cargar la identidad y poder hacer las
                        //-- transformaciones
      gluOrtho2D(0.0, 499.0, 0.0, 499.0)
}

void main(int argc, char** argv)
{
      /*  Inicializacion GLUT estándar*/
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      glutInitWindowSize(500, 500); /* ventana 500x500 pixeles */
      glutInitWindowPosition(0, 0);
      glutCreateWindow("Transformaciones con OpenGL");
      glutDisplayFunc(display);
      myinit(); /* fija o establece los atributos */
      glutMainLoop(); /* entra a un ciclo de evento */
}
```

## 2.2. Rotación.

```
void Rota()
{
      //--triangulo antes de rotar
      glColor3f(0.5f, 1.0f, 0.7f);
      glBegin(GL_TRIANGLES);
        glVertex2f(100.0f, 100.0f);
        glVertex2f(200.0f, 100.0f);
        glVertex2f(150.0f, 150.0f);
      glEnd();


      //--triangulo rotado
      glRotatef(45, 0, 0, 1);
      glBegin(GL_TRIANGLES);
        glVertex2f(100.0f, 100.0f);
        glVertex2f(200.0f, 100.0f);
        glVertex2f(150.0f, 150.0f);
      glEnd();
}
```

## 2.3. Escalamiento

```
void Escala()
{
      //--triangulo antes de escalar
      glColor3f(0.5f, 1.0f, 0.7f);
      glBegin(GL_TRIANGLES);
      glVertex2f(100.0f, 100.0f);
      glVertex2f(200.0f, 100.0f);
      glVertex2f(150.0f, 150.0f);
      glEnd();

    //--triangulo despues de escalar

      glScalef(2.0, 2.0, 2.0);
      glBegin(GL_TRIANGLES);
       glVertex2f(100.0f, 100.0f);
       glVertex2f(200.0f, 100.0f);
       glVertex2f(150.0f, 150.0f);
      glEnd();
}
```

## 2.4. Composición de transformaciones.

```
void TransfComp()
{
      glColor3f(0.5f, 1.0f, 0.7f);
      glBegin(GL_TRIANGLES);
      glVertex2f(100.0f, 100.0f);
      glVertex2f(200.0f, 100.0f);
      glVertex2f(150.0f, 150.0f);
      glEnd();


      glRotatef(45, 0, 0, 1);
      glTranslatef(200.0f, 200.0f, 0.0f);

      glBegin(GL_TRIANGLES);
        glVertex2f(100.0f, 100.0f);
        glVertex2f(200.0f, 100.0f);
        glVertex2f(150.0f, 150.0f);
```

```
        glEnd();
       glFlush();
 }
```

## 2.5. ANIMACION

```
#include <GL/glut.h>
#include<GL/glew.h>

// Global variable
GLfloat angle = 0.0f;  // Current rotational angle of the shapes

/* Initialize OpenGL Graphics */
void initGL() {
    // Set "clearing" or background color
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
}

/* Called back when there is no other event to be handled */
void idle() {
    glutPostRedisplay();  // Post a re-paint request to activate display()
}

/* Handler for window-repaint event. Call back when the window first
appears and
   whenever the window needs to be re-painted. */
void display() {
    glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer
    glMatrixMode(GL_MODELVIEW);     // To operate on Model-View matrix
    glLoadIdentity();               // Reset the model-view matrix

    glPushMatrix();                      // Save model-view matrix setting
    glTranslatef(-0.5f, 0.4f, 0.0f);    // Translate
    glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
    glBegin(GL_QUADS);                   // Each set of 4 vertices form a
quad
    glColor3f(1.0f, 0.0f, 0.0f);     // Red
    glVertex2f(-0.3f, -0.3f);
    glVertex2f(0.3f, -0.3f);
    glVertex2f(0.3f, 0.3f);
    glVertex2f(-0.3f, 0.3f);
    glEnd();
    glPopMatrix();                       // Restore the model-view matrix

    glPushMatrix();                      // Save model-view matrix setting
    glTranslatef(-0.4f, -0.3f, 0.0f);   // Translate
    glRotatef(angle, 0.0f, 0.0f, 1.0f); // rotate by angle in degrees
    glBegin(GL_QUADS);
    glColor3f(0.0f, 1.0f, 0.0f); // Green
    glVertex2f(-0.3f, -0.3f);
    glVertex2f(0.3f, -0.3f);
    glVertex2f(0.3f, 0.3f);
    glVertex2f(-0.3f, 0.3f);
    glEnd();
    glPopMatrix();  // Restore the model-view matrix

    glutSwapBuffers(); // Double buffered-swap the front and back buffers

    // Change the rotational angle after each display()
    angle += 0.2f;
}
```

```
/* Main function: GLUT runs as a console application starting at main() */
int main(int argc, char** argv) {
    glutInit(&argc, argv);  // Initialize GLUT
    glutInitDisplayMode(GLUT_DOUBLE);// Enable double buffered mode
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Animation via Idle Function");
    glutDisplayFunc(display);
    glutIdleFunc(idle);// Register callback handler if no other event
    initGL();           // Our own OpenGL initialization
    glutMainLoop();     // Enter the infinite event-processing loop
    return 0;
}
```

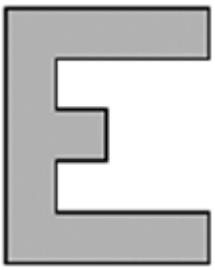
## 3. ACTIVIDADES EN CLASE

3.1. Escribir un programa que escale por 2 en ambas direcciones, luego rote 30° y finalmente traslade un triángulo en el vector (80,20).

3.2. Escribir un programa que traslade y luego rote un cuadrado: (poner los parámetros de traslación y ángulo de rotación a criterio)

## 4. TAREA.

4.1. Escribir un programa para realizar una animación de rotación de la siguiente figura.



## 1. 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.