INDEX

INTRODUCTION

This project aims to demonstrate the use of the BGI graphics library to animate an insect or bird-like figure flying across the screen. By combining various shapes and colors, the animation seeks to mimic wing movements and simulate flight. The animation progresses through loops and timed delays, creating an illusion of movement and flight.

THEORY

The animation presented in this program is a simple example of frame-by-frame animation, a concept widely used in both 2D and 3D computer graphics. In this case, we utilize the Borland Graphics Interface (BGI) functions such as `ellipse()`, `line()`, `circle()`, and `cleardevice()` to create the visual components of a flying insect or bird. These graphical primitives are combined and manipulated to form different parts of the insect, such as the head, body, wings, and other distinguishing features.

Frame-by-Frame Animation

At the heart of the animation is the principle of frame-by-frame redrawing, which mimics the way films or video games create motion. Instead of actual movement, individual frames are displayed in rapid succession, each slightly different from the previous one. When viewed in real-time, this creates the illusion of continuous motion. In this program, each frame represents the insect at a slightly different horizontal position on the screen, which makes it appear to be flying from one side of the screen to the other.

For every iteration of the loop, the position of the insect is incrementally changed by shifting its x-coordinates (`400 - i`, `478 - i`, etc.). This gradual shift gives the appearance of smooth motion. The BGI function `cleardevice()` is used after each frame is drawn to remove the previous image, preventing a trail effect and ensuring that only the current frame is displayed.

Shape Manipulation for Animation

The body and wings of the insect are drawn using ellipses and lines, which are fundamental shapes available in the BGI graphics library. The BGI function `ellipse()` allows you to draw ellipses by specifying the center coordinates, start and end angles, and radii along the x and y axes. In the animation, the body of the insect is represented by a series of ellipses with different dimensions, positioned to suggest various parts of the insect's anatomy, such as the head and torso.

The most important aspect of the animation is the simulation of wing flapping. This is achieved by altering the shape and angle of the ellipses used to represent the wings. The code alternates between two different wing positions depending on the value of the loop counter (`i`). When `i % 8 == 0`, the wings are drawn in one position, and in all other cases, they are drawn in an alternate position. By switching between these two states, the program creates the impression of the wings moving up and down as the insect "flies." This simple conditional logic introduces variability to the animation and enhances its realism.

Timing and Control

The `delay()` function, also from the BGI library, is used to control the speed of the animation. A delay of 35 milliseconds between each frame ensures that the animation runs at a smooth and visually pleasing pace. Without this delay, the frames would update too quickly, causing the animation to appear erratic or too fast for the human eye to follow. By carefully choosing the delay value, the program balances performance with realism.

The `kbhit()` function is used to detect if a key is pressed. As long as no key is pressed, the program continues running the animation in a loop, allowing the insect to fly continuously. When a key is pressed, the loop breaks, and the animation halts. This gives the user control over when to stop the program, ensuring it doesn’t run indefinitely.

Graphics and Computational Efficiency

Although the program provides a simple visual animation, it is important to note that it is computationally efficient. By using basic geometric shapes like ellipses, circles, and lines, the program keeps the complexity low. These shapes are drawn and erased quickly, minimizing the strain on the processor. Moreover, the use of a fixed increment for movement (`i++`) ensures that the insect moves at a steady pace across the screen without requiring complex calculations or additional memory overhead.

The concept of redrawing the scene after clearing the previous frame is a fundamental approach in game development and animations. While this particular implementation is basic, more advanced techniques, such as double buffering (where two screen buffers are used to avoid flickering), can be used to further improve the performance and visual quality of the animation.

OPERATION AND FUNCTIONING

- **Initialization:** The graphics mode is initialized with `initgraph()` using a specific path to the BGI driver.

- **Looping Structure:** The animation is drawn within a `for` loop, moving the figure across the screen incrementally.

- **Animation Details:** Two different wing positions are alternated to simulate a flapping effect. The `if` statement checks the loop variable `i` and switches the wing position to give the impression of movement.

- **Delay and Redrawing:** A delay of 35 milliseconds is used between each frame to control the speed of the animation, and `cleardevice()` clears the screen before the next frame is drawn.

- **Exit Condition:** The loop continues until a key is pressed, at which point `getch()` waits for user input to close the graphics window.

SYSTEM DESIGN

The program design consists of:

- **Main Animation Loop:** Runs the animation using a `for` loop.
- **Body Elements:** Ellipses, lines, and circles that form the insect’s head, wings, and body.
- **Conditional Wing Flapping:** Uses an `if` condition to alter wing positions at regular intervals, simulating movement.
- **Screen Clearing and Delay:** Utilizes `cleardevice()` for screen refresh and `delay()` to control frame timing.

REQUIREMENTS

- **Software:** Turbo C/C++ Compiler with the BGI graphics library.
- **Hardware:** Any compatible system that supports Turbo C and graphics mode.
- **Libraries:** `<graphics.h>`, `<conio.h>`, `<dos.h>`, `<stdlib.h>`, and `<stdio.h>`.
- **Additional Files:** The BGI files, such as `BGI` and `EGAVGA.BGI`, typically located in the Turbo C directory.

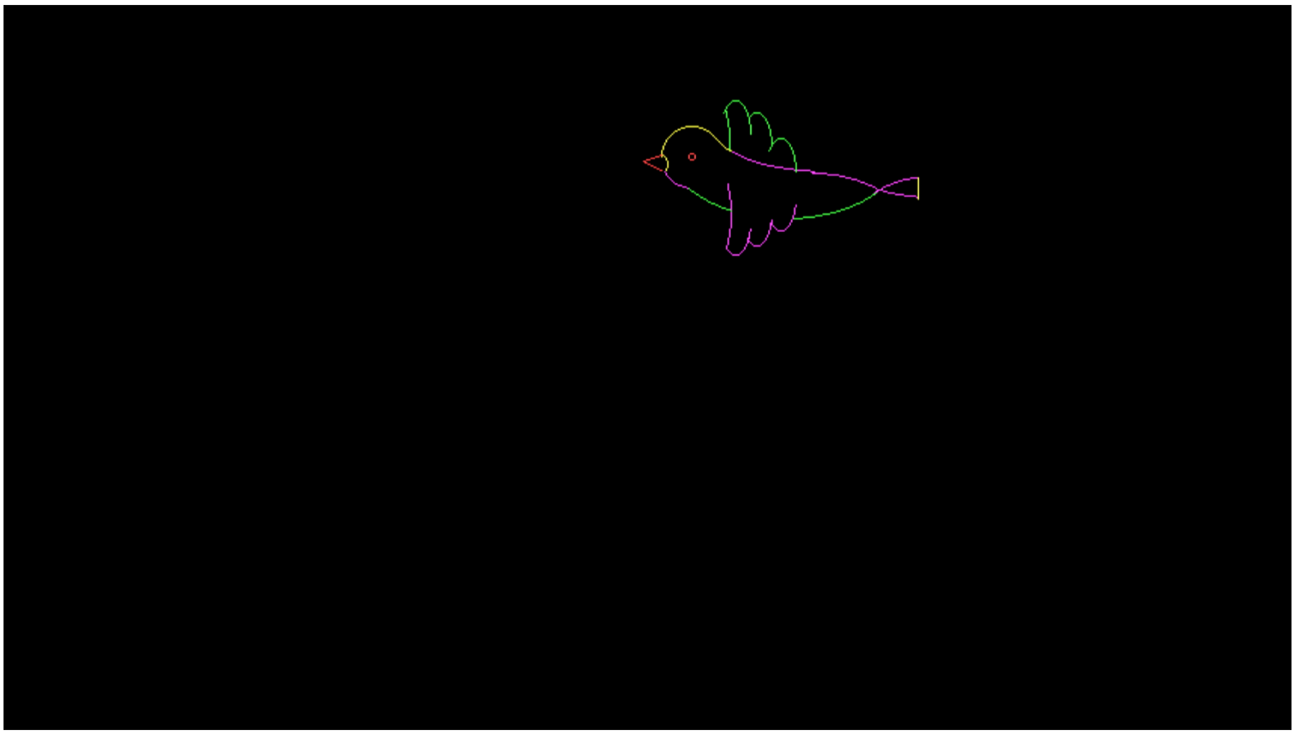
IMPLEMENTATION

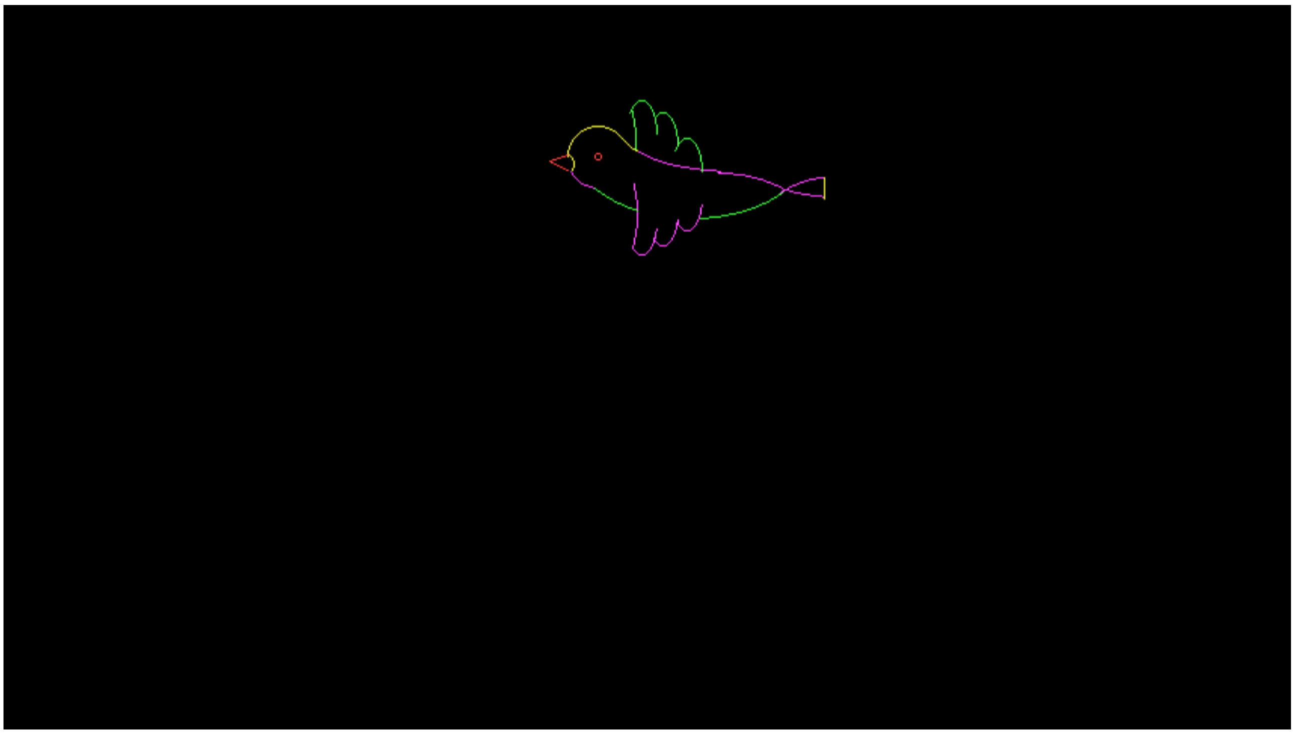
CODE:

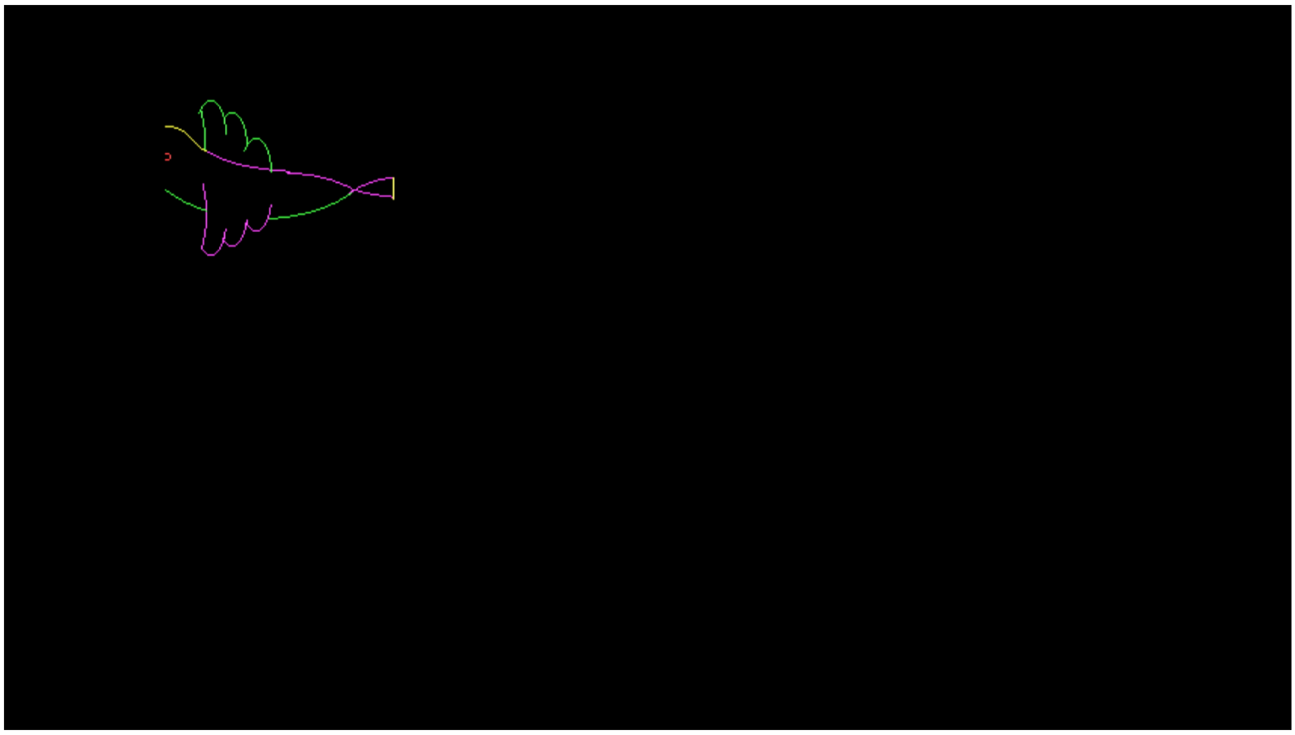
```
#include <graphics.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>

void main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

   while (!kbhit()) {
   int i;
   for (i = 0; i < 550; i++) {
         // Head
         setcolor(14);
         ellipse(400 - i, 100, 40, 180, 20, 20);
         ellipse(478 - i, 78, 199, 220, 66, 30);
         ellipse(377 - i, 105, 320, 70, 7, 7);

         // Back
         setcolor(12);
         line(380 - i, 99, 368 - i, 103);
         line(368 - i, 103, 380 - i, 109);
         setcolor(13);
         ellipse(435 - i, 95, 200, 250, 9, 4);
         ellipse(402 - i, 107, 190, 260, 20, 14);

         if (i % 8 == 0) {
            // Wings in position 1
            setcolor(10);
            ellipse(389 - i, 90, 340, 20, 45, 60);
            ellipse(439 - i, 85, 0, 140, 10, 22);
            ellipse(453 - i, 93, 0, 120, 10, 22);
            ellipse(469 - i, 110, 0, 140, 10, 22);
            ellipse(385 - i, 74, 340, 10, 40, 60);
            ellipse(428 - i, 83, 45, 122, 6, 22);
            ellipse(462 - i, 91, 216, 320, 80, 50);
         } else {
            // Wings in position 2
            ellipse(381 - i, 138, 340, 20, 45, 60);
            ellipse(429 - i, 143, 230, 350, 10, 22);
            ellipse(443 - i, 137, 230, 350, 10, 22);
            ellipse(459 - i, 127, 230, 350, 10, 22);
            ellipse(494 - i, 90, 207, 260, 70, 20);
```

```
                setcolor(10);
                ellipse(380 - i, 90, 357, 20, 45, 60);
                ellipse(429 - i, 85, 0, 140, 10, 22);
                ellipse(443 - i, 93, 0, 120, 10, 22);
                ellipse(459 - i, 110, 0, 140, 10, 22);
                ellipse(462 - i, 91, 216, 244, 80, 50);
                ellipse(462 - i, 91, 275, 320, 80, 50);
            }

            // Body and other parts
            setcolor(12);
            circle(400 - i, 100, 2);
            setcolor(13);
            ellipse(480 - i, 161, 50, 90, 70, 50);
            ellipse(560 - i, 143, 100, 140, 50, 30);
            ellipse(580 - i, 118, 200, 240, 60, 10);

            setcolor(14);
            line(550 - i, 114, 550 - i, 128);

            delay(35);
            cleardevice();
        }
    }

    getch();
    closegraph();
}
```

OUTPUT:







CONCLUSION

The program successfully demonstrates basic graphical animation in C using the BGI graphics library. By combining simple shapes and position alternation, the code simulates an insect or bird in flight. The program emphasizes frame-based animation and showcases how graphics libraries can enhance C programming with visual elements.

REFERENCES

1. Borland Graphics Interface (BGI) Documentation
https://www.tutorialspoint.com/c_standard_library/c_function_graphics.h.htm

2. C Graphics Programming
https://www.geeksforgeeks.org/graphics-in-c/

3. Turbo C/C++ Compiler Overview
https://www.tutorialspoint.com/turbo_c/index.htm

4. Graphics Programming in C
https://www.codeproject.com/Articles/746130/Graphics-in-C-Using-BGI

5. C Programming Language Reference
https://www.learn-c.org/