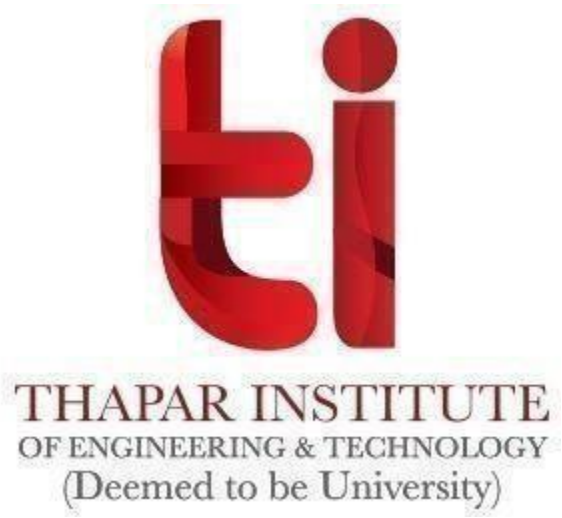
COMPUTER GRAPHICS UCS505

Project Title: Cathode Ray Tube Working

Submitted By:
Ramneet Singh (102003071)
Gaurav Pahwa (102003087)
Kunal Demla (102003088)

Group: 3COE4
BE Third Year - COE

Submitted To:
Dr. Anupam Garg



Computer Science and Engineering Department
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA

Table of Contents

Introduction

Cathode rays are streams of electrons emitted by the negatively charged electrode, or cathode, in a vacuum tube. They were first discovered in the late 19th century by scientists studying the behavior of electric current in evacuated glass tubes. The study of cathode rays led to many important discoveries in physics, including identifying electrons as fundamental particles, developing television and computer screens, and inventing X-ray technology. Cathode ray tubes have largely been replaced by newer technologies, but they remain an important part of the history of science and technology.

Our group chose to work on this project because it is an excellent opportunity to apply our knowledge of computer graphics and algorithms to create a simulation that is both educational and visually engaging. Our goal was to create an animation that accurately simulates the movement of electrons in a CRT and showcases the efficiency of the algorithms used in its implementation. We used programming tools such as C++ with OpenGL to develop the simulation. Owing to that, we will be presenting our project using the basic functions of OpenGL. The project is developed using the "OpenGL" language, "Cathode Ray Tube Working Animation".

Our project is a visually stimulating demonstration of the inner workings of a cathode ray tube. Beginning with constructing the fundamental tube frame, we depict the intricate electron emission process from the heated cathode. Through dynamic illustrations, viewers can observe the directional flow of these electrons as they travel through the horizontal and vertical plates, culminating in a mesmerizing display of light and color. This project provides an immersive and educational experience that highlights the complexity and beauty of this significant technological advancement.

This mini project is implemented simply, and we have used user-defined and inbuilt functions of OpenGL.

Computer Graphics Concepts

The following concepts were used to build the project from scratch:

1. **For line drawing** - Bresenham's line drawing algorithm
2. **For circle drawing** - Bresenham's circle drawing algorithm
3. **For color filling** - Flood-fill color filling algorithm
4. **For plotting points** - OpenGL inbuilt function
5. **For Calculating Transformation** - Manually
6. **For Motion calculation** - 2D Transformation
7. **OpenGL**- A cross-platform graphics API for creating 2D and 3D applications.
8. **Creation of shapes** - Primitives such as GL_POLYGON
9. **For Ellipse drawing -** Midpoint ellipse drawing algorithm
10. **For delay in animation** - Sleep() function
11. **For timer updation** - glutTimerFunc()
12. **For Interactions** - glutMenu()
13. **For window parameter tuning** - myinit()

User Defined Functions

1. **void display():** This function is the main display function that creates the whole animation. It initially wireframes the whole structure and then fills color in the same.
2. **void bresenham():** This function takes 4 inputs for points for the line to be drawn.
3. **void drawEllipse():** This function takes 4 inputs and draws the ellipse.
4. **void floodfill():** This function takes 4 inputs for starting point, initial color, and new color and colors the surrounding area.
5. **void bresenhamcircle():** This function takes 3 inputs and draws the circle.
6. **void drawArrow():** This draws, translates, and colors the electron paths on the diagram through the tube.
7. **void drawVerticleArrow():** This draws the magnetic field arrows on the strips.
8. **void updateElapsedTime():** Updates and keeps checking the total time elapsed and changes flags for function calls.
9. **void timer():** Changes colors for blinking arrows.
10. **void displayElectrons():** Shows the fluorescence effect displayed on the screen with visual bright spot patterns.
11. **void updateArrowLevels():** Keeps the count of arrows drawn.
12. **void displayWave():** Shows the output screen with the waveform on electrons on the CRT.
13. **void menu():** Calls the correct function for the chosen menu item.

OpenGL CODE

```
/*

 Computer Graphics Project (Semester 6)

Title: Cathode Ray Tube working animation
Member 1: Ramneet Singh (102003071)
Member 2: Gaurav Pahwa (102003087)
Member 3: Kunal Demla (102003088)
Group: 3COE4
Date: April 2023

 */


#include <GL/glut.h>
#include <windows.h>
#include <bits/stdc++.h>
#include <stack>
#include <utility>
#include<iostream>
using namespace std;

int width = 1000;
int height = 500;
float br = 0.81;
float bg = 0.71;
float bb = 0.65;
float delay = 1;
void drawEllipse(int a, int b, int xc, int yc) {
   glClear(GL_COLOR_BUFFER_BIT);

   float x = 0, y = b;
   float d1 = (b * b) - (a * a * b) + (0.25 * a * a);
   float dx = 2 * b * b * x;
   float dy = 2 * a * a * y;

   while (dx < dy) {
      glBegin(GL_POINTS);
      glVertex2f(x + xc, y + yc);
      glVertex2f(-x + xc, y + yc);
      glVertex2f(x + xc, -y + yc);
      glVertex2f(-x + xc, -y + yc);
      glEnd();
```

```
            glFlush();
            Sleep(delay);
            x++;
            dx = dx + (2 * b * b);
            if (d1 < 0) {
                d1 = d1 + dx + (b * b);
            }
            else {
                y--;
                dy = dy - (2 * a * a);
                d1 = d1 + dx - dy + (b * b);
            }
        }

        float d2 = ((b * b) * ((x + 0.5) * (x + 0.5))) +
            ((a * a) * ((y - 1) * (y - 1))) -
            (a * a * b * b);

        while (y >= 0) {
            glBegin(GL_POINTS);
            glVertex2f(x + xc, y + yc);
            glVertex2f(-x + xc, y + yc);
            glVertex2f(x + xc, -y + yc);
            glVertex2f(-x + xc, -y + yc);
            glEnd();
            glFlush();
            Sleep(delay);
            y--;
            dy = dy - (2 * a * a);
            if (d2 > 0) {
                d2 = d2 - dy + (a * a);
            }
            else {
                x++;
                dx = dx + (2 * b * b);
                d2 = d2 + dx - dy + (a * a);
            }
        }
    }

    void bresenham(int x1, int y1, int x2, int y2) {
        int dx = x2 - x1;
        int dy = y2 - y1;
        int xinc = 1, yinc = 1;
        if (dx < 0) xinc = -1;
        if (dy < 0) yinc = -1;
        dx = abs(dx);
```

```
dy = abs(dy);
int dy2 = 2 * dy;
int dydx2 = 2 * dy - 2 * dx;
int dx2 = 2 * dx;
int dxdy2 = 2 * dx - 2 * dy;
int x = x1;
int y = y1;
int p;
if (dx > dy) {
   p = 2 * dy - dx;
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   for (int i = 0; i < dx; i++) {
      if (p < 0) {
         p = p + dy2;
         x = x + xinc;
         glBegin(GL_POINTS);
         glVertex2i(x, y);
         glEnd();
         glFlush();
         Sleep(delay);
      }
      else {
         p = p + dydx2;
         x = x + xinc;
         y = y + yinc;
         glBegin(GL_POINTS);
         glVertex2i(x, y);
         glEnd();
         glFlush();
         Sleep(delay);
      }
   }
}
else {
   p = 2 * dx - dy;
   glBegin(GL_POINTS);
   glVertex2i(x, y);
   for (int i = 0; i < dy; i++) {
      if (p < 0) {
         p = p + dx2;
         y = y + yinc;
         glBegin(GL_POINTS);
         glVertex2i(x, y);
         glEnd();
         glFlush();
         Sleep(delay);
```

```
            }
            else {
               p = p + dxdy2;
               x = x + xinc;
               y = y + yinc;
               glBegin(GL_POINTS);
               glVertex2i(x, y);
               glEnd();
               glFlush();
               Sleep(delay);
            }
         }
         glEnd();
      }
   }

   void bresenhamWithouDelay(int x1, int y1, int x2, int y2) {
      int dx = x2 - x1;
      int dy = y2 - y1;
      int xinc = 1, yinc = 1;
      if (dx < 0) xinc = -1;
      if (dy < 0) yinc = -1;
      dx = abs(dx);
      dy = abs(dy);
      int dy2 = 2 * dy;
      int dydx2 = 2 * dy - 2 * dx;
      int dx2 = 2 * dx;
      int dxdy2 = 2 * dx - 2 * dy;
      int x = x1;
      int y = y1;
      int p;
      if (dx > dy) {
         p = 2 * dy - dx;
         glBegin(GL_POINTS);
         glVertex2i(x, y);
         glEnd();
         for (int i = 0; i < dx; i++) {
            if (p < 0) {
               p = p + dy2;
               x = x + xinc;
               glBegin(GL_POINTS);
               glVertex2i(x, y);
               glEnd();
            }
            else {
               p = p + dydx2;
               x = x + xinc;
```

```
            y = y + yinc;
            glBegin(GL_POINTS);
            glVertex2i(x, y);
            glEnd();
         }
      }
   }
   else {
      p = 2 * dx - dy;
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      for (int i = 0; i < dy; i++) {
         if (p < 0) {
            p = p + dx2;
            y = y + yinc;
            glBegin(GL_POINTS);
            glVertex2i(x, y);
            glEnd();
         }
         else {
            p = p + dxdy2;
            x = x + xinc;
            y = y + yinc;
            glBegin(GL_POINTS);
            glVertex2i(x, y);
            glEnd();
         }
      }
      glFlush();
   }
}

void bresenhamcircle(int xc, int yc, int r) {
   int p = 3 - 2 * r;
   int x = 0, y = r;
   glBegin(GL_POINTS);
   glVertex2i(x + xc, y + yc);
   glVertex2i(x + xc, -y + yc);
   glVertex2i(y + xc, x + yc);
   glVertex2i(-y + xc, x + yc);
   glEnd();
   while (y > x) {
      x = x + 1;
      if (p <= 0) {
         p = p + 4 * x + 6;
      }
      else {
```

```
            p = p + 4 * x - 4 * y + 10;
            y--;
        }
        if (y < x)
            break;
        glBegin(GL_POINTS);
        glVertex2i(x + xc, y + yc);
        glVertex2i(-x + xc, y + yc);
        glVertex2i(x + xc, -y + yc);
        glVertex2i(-x + xc, -y + yc);
        glVertex2i(y + xc, x + yc);
        glVertex2i(-y + xc, x + yc);
        glVertex2i(y + xc, -x + yc);
        glVertex2i(-y + xc, -x + yc);
        glEnd();
        glFlush();
        Sleep(delay);
    }
}

float twoPlaces(float a) {
    return round(a * 100.0) / 100.0;
}

void floodFill(int x, int y, float* fillColor, float* interiorColor) {
    float currentColor[3];
    glReadPixels(x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, currentColor);

    std::stack<std::pair<int, int>> pixels;
    pixels.push(std::make_pair(x, y));
    while (!pixels.empty()) {
        std::pair<int, int> currentPixel = pixels.top();
        pixels.pop();
        int cx = currentPixel.first;
        int cy = currentPixel.second;
        glReadPixels(cx, cy, 1.0, 1.0, GL_RGB, GL_FLOAT, currentColor);
        //std::cout<<twoPlaces(currentColor[0])<<" "<<interiorColor[0]<<std::endl;
        if (twoPlaces(currentColor[0]) == interiorColor[0] && twoPlaces(currentColor[1]) ==
interiorColor[1] && twoPlaces(currentColor[2]) == interiorColor[2]) {
            glBegin(GL_POINTS);
            glColor3fv(fillColor);
            glVertex2i(cx, cy);
            glEnd();
            glFlush();
            pixels.push(std::make_pair(cx + 1, cy));
            pixels.push(std::make_pair(cx - 1, cy));
            pixels.push(std::make_pair(cx, cy + 1));
```

```
            pixels.push(std::make_pair(cx, cy - 1));
        }
    }
}


bool isBlinking = true;
int blinkDuration = 500; // Blink duration in milliseconds
int elapsedTime = 0; // Elapsed time in milliseconds
int pending = 1;
int arrowUpdatePending = 1;
int arrowLevel = 0;

void drawArrow() {
    if (isBlinking) {

        for (int i = 0; i <= arrowLevel; i++) {
        int x = i * 50;
        glColor3f(0.5, 0.5, 0.5);// grey
        if (i > 9 && i < 13) continue;
        bresenhamWithouDelay(175 + x, 255, 195 + x, 255);
        bresenhamWithouDelay(195 + x, 255, 195 + x, 257);
        bresenhamWithouDelay(195 + x, 257, 207 + x, 250);
        bresenhamWithouDelay(207 + x, 250, 195 + x, 243);
        bresenhamWithouDelay(195 + x, 243, 195 + x, 245);
        bresenhamWithouDelay(195 + x, 245, 175 + x, 245);
        bresenhamWithouDelay(175 + x, 245, 175 + x, 255);

        //heater color change
        glColor3f(0.8,0.33,0.0);
        glBegin(GL_POLYGON);
        glVertex2f(55, 260);
        glVertex2f(95, 260);
        glVertex2f(95, 240);
        glVertex2f(55, 240);
        glEnd();
        if (arrowLevel>9){
            glColor3f(0.0,0.69,0.42);
            glBegin(GL_POLYGON);
        glVertex2f(673-2, 297+2);
        glVertex2f(757+1, 297+2);
        glVertex2f(757+1, 253-1);
        glVertex2f(673-2, 253-1);
        glEnd();
            glBegin(GL_POLYGON);
        glVertex2f(670+1, 252-2);
        glVertex2f(770, 252-2);
```

```
    glVertex2f(770, 202+1);
    glVertex2f(670+1, 202+1);
    glEnd();

    glBegin(GL_POLYGON);
    glVertex2f(761, 302);
    glVertex2f(758, 252);
    glVertex2f(820, 265);
    glVertex2f(820, 315);
    glEnd();

    glBegin(GL_POLYGON);
    glVertex2f(771, 250);
    glVertex2f(768, 200);
    glVertex2f(830, 187);
    glVertex2f(830, 237);
    glEnd();
    }

}
}
else {

    for (int i = 0; i <= arrowLevel; i++) {
    int x = i * 50;
    glColor3f(0.56, 0.93, 0.56);//light green
    if (i > 9 && i < 13) continue;
    bresenhamWithouDelay(175 + x, 255, 195 + x, 255);
    bresenhamWithouDelay(195 + x, 255, 195 + x, 257);
    bresenhamWithouDelay(195 + x, 257, 207 + x, 250);
    bresenhamWithouDelay(207 + x, 250, 195 + x, 243);
    bresenhamWithouDelay(195 + x, 243, 195 + x, 245);
    bresenhamWithouDelay(195 + x, 245, 175 + x, 245);
    bresenhamWithouDelay(175 + x, 245, 175 + x, 255);

    //heater color change
    glColor3f(1.0,0.67,0.1);//bright orange
    glBegin(GL_POLYGON);
    glVertex2f(55, 260);
    glVertex2f(95, 260);
    glVertex2f(95, 240);
    glVertex2f(55, 240);
    glEnd();
    if (arrowLevel>9){
          glColor3f(0.69,0.42,0.0);
       glBegin(GL_POLYGON);
    glVertex2f(673-2, 297+2);
```

```
        glVertex2f(757+1, 297+2);
        glVertex2f(757+1, 253-1);
        glVertex2f(673-2, 253-1);
        glEnd();
           glBegin(GL_POLYGON);
        glVertex2f(670+1, 252-2);
        glVertex2f(770, 252-2);
        glVertex2f(770, 202+1);
        glVertex2f(670+1, 202+1);
        glEnd();

        glBegin(GL_POLYGON);
        glVertex2f(761, 302);
        glVertex2f(758, 252);
        glVertex2f(820, 265);
        glVertex2f(820, 315);
        glEnd();

        glBegin(GL_POLYGON);
        glVertex2f(771, 250);
        glVertex2f(768, 200);
        glVertex2f(830, 187);
        glVertex2f(830, 237);
        glEnd();
        }
    }
    }

    if (arrowLevel == 9) arrowLevel += 3;


}

void drawVerticalArrow() {
    if (isBlinking) {
        glColor3f(0.5, 0.5, 0.5);
        for (int i = 0; i < 2; i++) {
            int x = i * 20;
            bresenhamWithouDelay(478 + x, 205, 485 + x, 217);
            bresenhamWithouDelay(485 + x, 217, 492 + x, 205);
            bresenhamWithouDelay(485 + x, 217, 485 + x, 185);
        }
        glColor3f(0.0,1.0,1.0);
        glBegin(GL_POLYGON);
        glVertex2f(470, 265);
        glVertex2f(470, 275);
        glVertex2f(550, 275);
```

```
    glVertex2f(550, 265);
    glEnd();

     glBegin(GL_POLYGON);
    glVertex2f(540, 262);
    glVertex2f(535, 272);
    glVertex2f(595, 310);
    glVertex2f(600, 300);
    glEnd();

     glBegin(GL_POLYGON);
    glVertex2f(470, 235);
    glVertex2f(470, 225);
    glVertex2f(550, 225);
    glVertex2f(550, 235);
    glEnd();

    glBegin(GL_POLYGON);
    glVertex2f(540, 238);
    glVertex2f(535, 228);
    glVertex2f(595, 190);
    glVertex2f(600, 200);
    glEnd();
}
else {
    glColor3f(0.56, 0.93, 0.56);
    for (int i = 0; i < 2; i++) {
        int x = i * 20;
        bresenhamWithouDelay(478 + x, 205, 485 + x, 217);
        bresenhamWithouDelay(485 + x, 217, 492 + x, 205);
        bresenhamWithouDelay(485 + x, 217, 485 + x, 185);
    }
    glColor3f(1.0,0.0,1.0);
    glBegin(GL_POLYGON);
    glVertex2f(470, 265);
    glVertex2f(470, 275);
    glVertex2f(550, 275);
    glVertex2f(550, 265);
    glEnd();

     glBegin(GL_POLYGON);
    glVertex2f(540, 262);
    glVertex2f(535, 272);
    glVertex2f(595, 310);
    glVertex2f(600, 300);
    glEnd();
```

```
        glBegin(GL_POLYGON);
        glVertex2f(470, 235);
        glVertex2f(470, 225);
        glVertex2f(550, 225);
        glVertex2f(550, 235);
        glEnd();

        glBegin(GL_POLYGON);
        glVertex2f(540, 238);
        glVertex2f(535, 228);
        glVertex2f(595, 190);
        glVertex2f(600, 200);
        glEnd();
    }
}

void updateElapsedTime(int value)
{
    //std::cout<<elapsedTime<<std::endl;
    // Update elapsed time
    elapsedTime += blinkDuration;
    if (elapsedTime >= 20000)
        arrowUpdatePending = 0;
    if (elapsedTime >= 28000) {
        pending = 0;
    }
    else {
        // Set timer for next elapsed time update
        glutTimerFunc(blinkDuration, updateElapsedTime, 1);
    }
}

void timer(int value)
{
    if (!pending) {
        return;
    }
    // Toggle blinking on and off
    isBlinking = !isBlinking;

    // Set timer for next blink
    glutTimerFunc(blinkDuration, timer, 0);

    drawArrow();
    if (arrowLevel > 6)
        drawVerticalArrow();
}
```

```
void displayElectrons() {
   if (!arrowUpdatePending) {
      glPointSize(10.0);
      glColor3f(0.0, 1.0, 0.0);
      glBegin(GL_POINTS);
      glVertex2f(920.0, 250.0);
      glVertex2f(920.0, 230.0);
      glVertex2f(920.0, 270.0);
      glVertex2f(920.0, 210.0);
      glVertex2f(920.0, 290.0);
      glVertex2f(918.0, 190.0);
      glVertex2f(918.0, 310.0);
      glVertex2f(918.0, 170.0);
      glVertex2f(918.0, 330.0);
      glVertex2f(916.0, 150.0);
      glVertex2f(916.0, 350.0);
      glVertex2f(916.0, 130.0);
      glVertex2f(916.0, 370.0);
      glVertex2f(912.0, 110.0);
      glVertex2f(912.0, 390.0);
      glVertex2f(910.0, 90.0);
      glVertex2f(910.0, 410.0);
      glEnd();
      glFlush();
      glPointSize(2.0);
   }
}
void updateArrowLevel(int value)
{
   // points over the ellipse to be printed at the end
   if (!arrowUpdatePending) {
      displayElectrons();
      return;
   }
   arrowLevel++;
   // Set timer for next blink
   glutTimerFunc(2000, updateArrowLevel, 2);
}


void display() {
   // your drawing code goes here
   glClear(GL_COLOR_BUFFER_BIT);
   glPointSize(2.0);
   glLineWidth(3.0f);
   glColor3f(0.0, 0.0, 0.0);
```

```
//front ellipse
drawEllipse(25, 180, 900, 250);


//slanting lines
bresenham(725, 325, 900, 430);
bresenham(725, 175, 900, 70);

//box
bresenham(725, 325, 725, 175);
bresenham(725, 175, 50, 175);
bresenham(50, 175, 50, 325);
bresenham(50, 325, 725, 325);


//slim box
bresenham(50, 175, 40, 175);
bresenham(40, 175, 40, 325);
bresenham(40, 325, 50, 325);

//pins
bresenham(30, 305, 40, 305);
bresenham(30, 295, 40, 295);
bresenham(30, 205, 40, 205);
bresenham(30, 195, 40, 195);

//heater
bresenham(55, 260, 95, 260);
bresenham(95, 260, 95, 240);
bresenham(95, 240, 55, 240);
bresenham(55, 240, 55, 260);

// elecron balls
bresenhamcircle(105, 250, 10);
bresenhamcircle(125, 250, 10);
bresenhamcircle(145, 250, 10);

//ballcover
bresenham(95, 270, 160, 270);
bresenham(160, 270, 160, 230);
bresenham(160, 230, 95, 230);

//grid
bresenham(160, 280, 210, 280);
bresenham(210, 280, 210, 260);
bresenham(210, 240, 210, 220);
bresenham(210, 220, 160, 220);
```

```
// pre-accelerating node
bresenham(220, 270, 220, 280);
bresenham(220, 280, 255, 280);
bresenham(255, 280, 255, 270);
bresenham(255, 280, 290, 280);
bresenham(290, 280, 290, 270);

bresenham(220, 230, 220, 220);
bresenham(220, 220, 255, 220);
bresenham(255, 220, 255, 230);
bresenham(255, 220, 290, 220);
bresenham(290, 220, 290, 230);

// focusing node
bresenham(300, 270, 300, 280);
bresenham(300, 280, 335, 280);
bresenham(335, 280, 335, 270);
bresenham(335, 280, 370, 280);
bresenham(370, 280, 370, 270);

bresenham(300, 230, 300, 220);
bresenham(300, 220, 335, 220);
bresenham(335, 220, 335, 230);
bresenham(335, 220, 370, 220);
bresenham(370, 220, 370, 230);

// accelerating node
bresenham(380, 270, 380, 280);
bresenham(380, 280, 415, 280);
bresenham(415, 280, 415, 270);
bresenham(415, 280, 450, 280);
bresenham(450, 280, 450, 270);

bresenham(380, 230, 380, 220);
bresenham(380, 220, 415, 220);
bresenham(415, 220, 415, 230);
bresenham(415, 220, 450, 220);
bresenham(450, 220, 450, 230);

//vertical deflection plates
bresenham(470, 265, 470, 275);
bresenham(470, 275, 550, 275);
bresenham(550, 275, 550, 265);
bresenham(550, 265, 470, 265);

bresenham(540, 262, 535, 272);
```

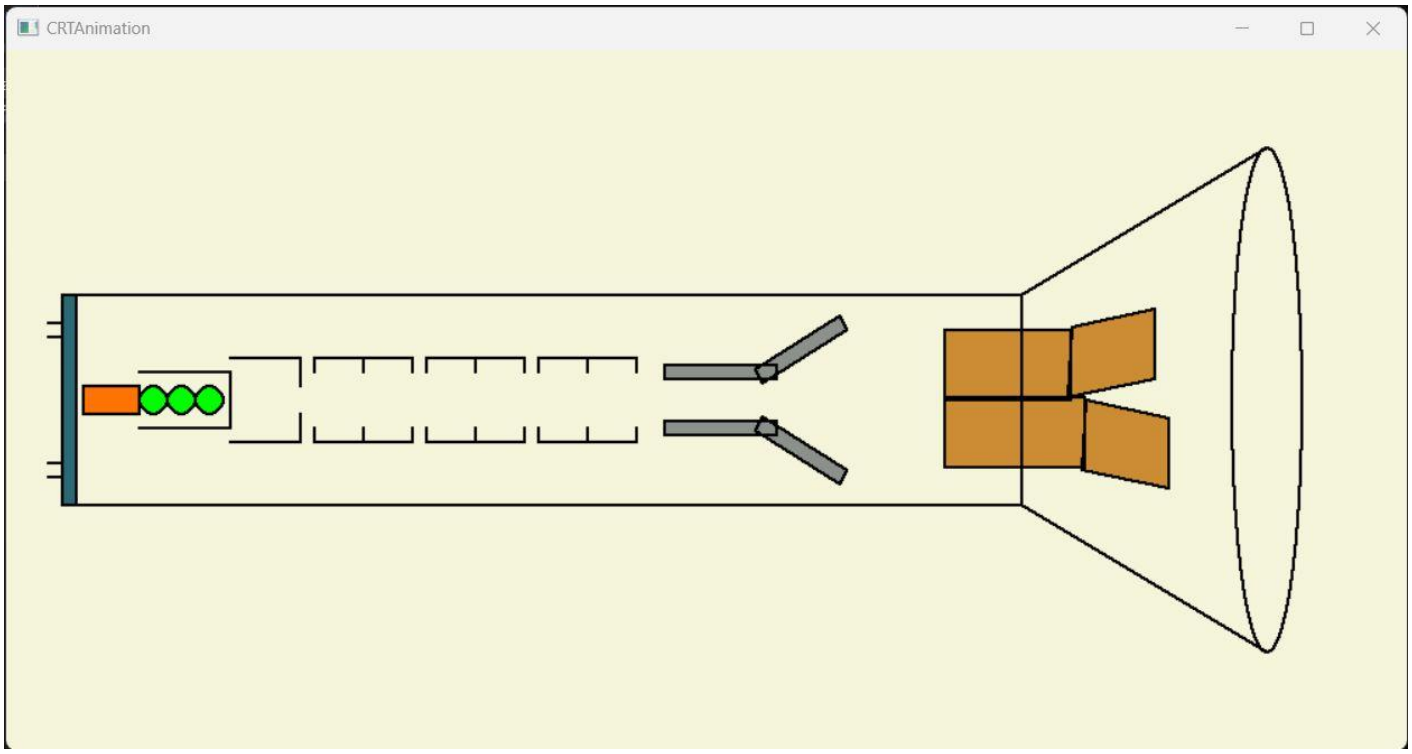
```
bresenham(535, 272, 595, 310);
bresenham(595, 310, 600, 300);
bresenham(600, 300, 540, 262);

bresenham(470, 235, 470, 225);
bresenham(470, 225, 550, 225);
bresenham(550, 225, 550, 235);
bresenham(550, 235, 470, 235);

bresenham(540, 238, 535, 228);
bresenham(535, 228, 595, 190);
bresenham(595, 190, 600, 200);
bresenham(600, 200, 540, 238);


//Horizontal deflection plates
bresenham(670, 300, 760, 300);
bresenham(760, 300, 760, 250);
bresenham(760, 250, 670, 250);
bresenham(670, 250, 670, 300);

bresenham(670, 252, 770, 252);
bresenham(770, 252, 770, 202);
bresenham(770, 202, 670, 202);
bresenham(670, 202, 670, 252);

bresenham(761, 302, 758, 252);
bresenham(758, 252, 820, 265);
bresenham(820, 265, 820, 315);
bresenham(820, 315, 761, 302);

bresenham(771, 250, 768, 200);
bresenham(768, 200, 830, 187);
bresenham(830, 187, 830, 237);
bresenham(830, 237, 771, 250);


glPointSize(1.0);
//color it

//color strip
float newColor[3] = { 0.0, 1.0, 1.0 }; // cyan
float bgColor[3] = { br,bg,bb }; // wood
floodFill(41, 300, newColor, bgColor);


//color heater
```

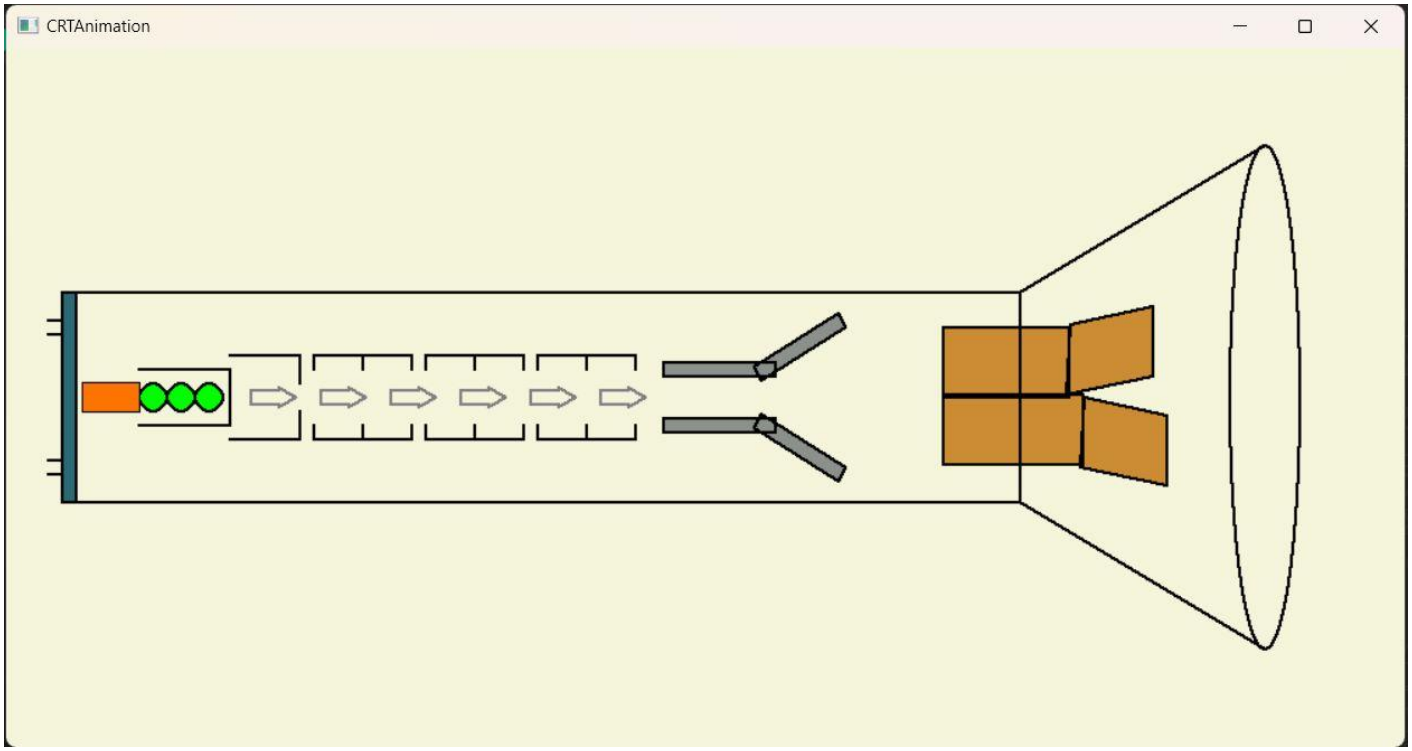
```
    newColor[0] = 0.88; newColor[1] = 0.34; newColor[2] = 0.13;
    bgColor[0] = br;   bgColor[1] = bg; bgColor[2] = bb;
    floodFill(56, 250, newColor, bgColor);
    //color electrons
    newColor[0] = 0.0;
    newColor[1] = 1.0;
    newColor[2] = 0.0;
    floodFill(105, 250, newColor, bgColor);
    floodFill(125, 250, newColor, bgColor);
    floodFill(145, 250, newColor, bgColor);

    //vertical defection plates
    newColor[0] = 0.0;
    newColor[1] = 1.0;
    newColor[2] = 1.0;
    floodFill(471, 266, newColor, bgColor);
    floodFill(545, 270, newColor, bgColor);
    floodFill(590, 295, newColor, bgColor);
    floodFill(475, 230, newColor, bgColor);
    floodFill(545, 230, newColor, bgColor);
    floodFill(590, 195, newColor, bgColor);

    //horizontal deflection plates
    newColor[0] = 1.0;
    newColor[1] = 0.65;
    newColor[2] = 0.0;
    floodFill(673, 290, newColor, bgColor);
    floodFill(730, 290, newColor, bgColor);
    floodFill(673, 240, newColor, bgColor);
    floodFill(730, 240, newColor, bgColor);
    floodFill(765, 270, newColor, bgColor);
    floodFill(775, 230, newColor, bgColor);
    glPointSize(2.0);
    glFlush();
}

void displayWave(float x,float y,float h){
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POLYGON);
    glVertex2i(x-5,y+h+5);
    glVertex2i(x+185,y+h+5);
    glVertex2i(x+185,y-h-5);
    glVertex2i(x-5,y-h-5);
    glEnd();
    glFlush();
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_LINES);
```

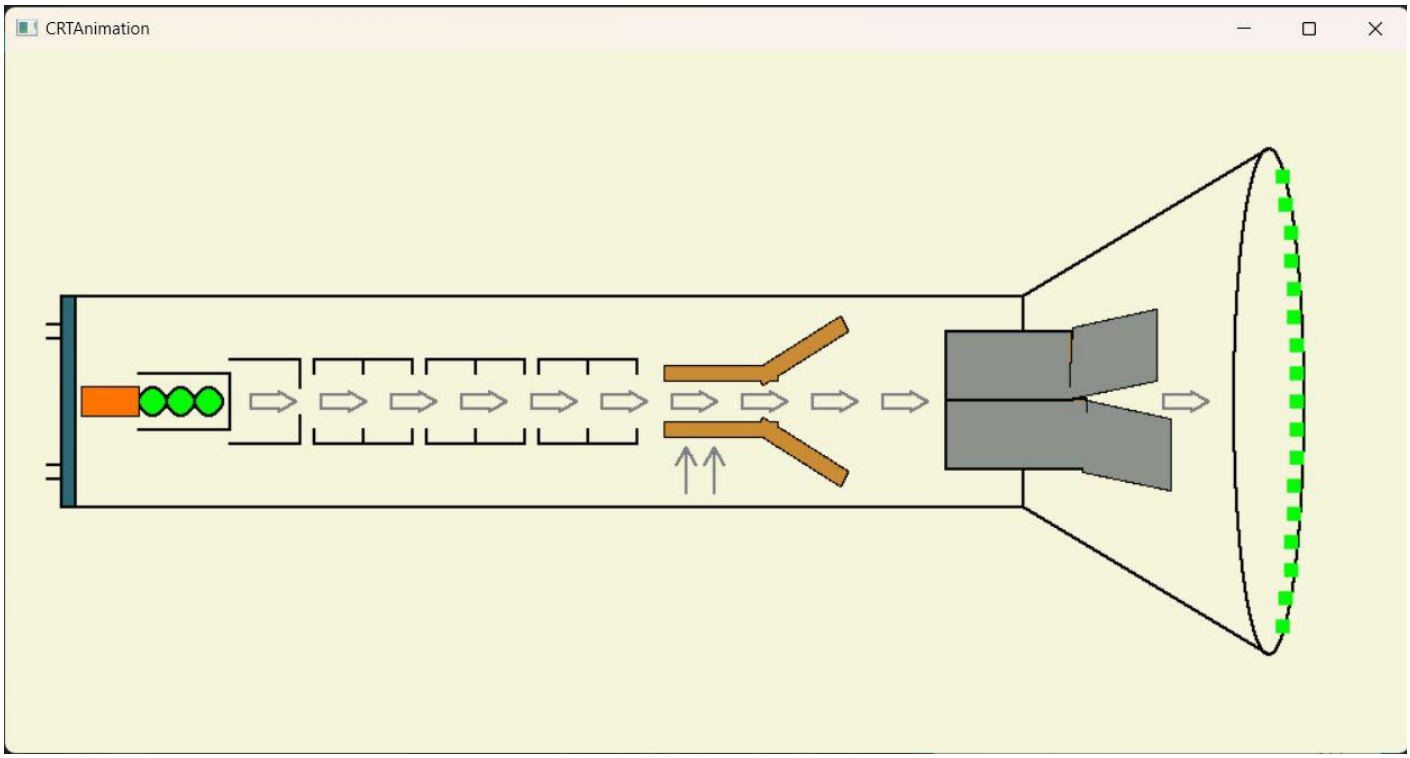
```
    glVertex2i(x-5,y+h+5);
    glVertex2i(x+185,y+h+5);
    glVertex2i(x+185,y+h+5);
    glVertex2i(x+185,y-h-5);
    glVertex2i(x+185,y-h-5);
    glVertex2i(x-5,y-h-5);
    glVertex2i(x-5,y-h-5);
    glVertex2i(x-5,y+h+5);
    glEnd();
    glFlush();
    glBegin(GL_LINES);
    glLineWidth(0.5);
    glVertex2i(x-5,y);
    glVertex2i(x+185,y);
    glEnd();
    glFlush();

    while(true){
        glColor3f(0.0,1.0,0.5);
        for(int i=0; i<=180; i++){
            float temp = sin(((float)i)*3.1415/90);
            temp*=h;
            glBegin(GL_POINTS);
            glVertex2f(x+i,y+temp);
            glEnd();
            glFlush();
            Sleep(delay);
        }
        glColor3f(1.0,0.0,0.5);
        for(int i=0; i<=180; i++){
            float temp = sin(((float)i)*3.1415/90);
            temp*=h;
            glBegin(GL_POINTS);
            glVertex2f(x+i,y+temp);
            glEnd();
            glFlush();
            Sleep(delay);
        }
    }
}

void menu(int id) {
    switch (id) {
    case 1:
        glutTimerFunc(blinkDuration, timer, 0);
        glutTimerFunc(blinkDuration, updateElapsedTime, 1);
        glutTimerFunc(2000, updateArrowLevel, 2);
```

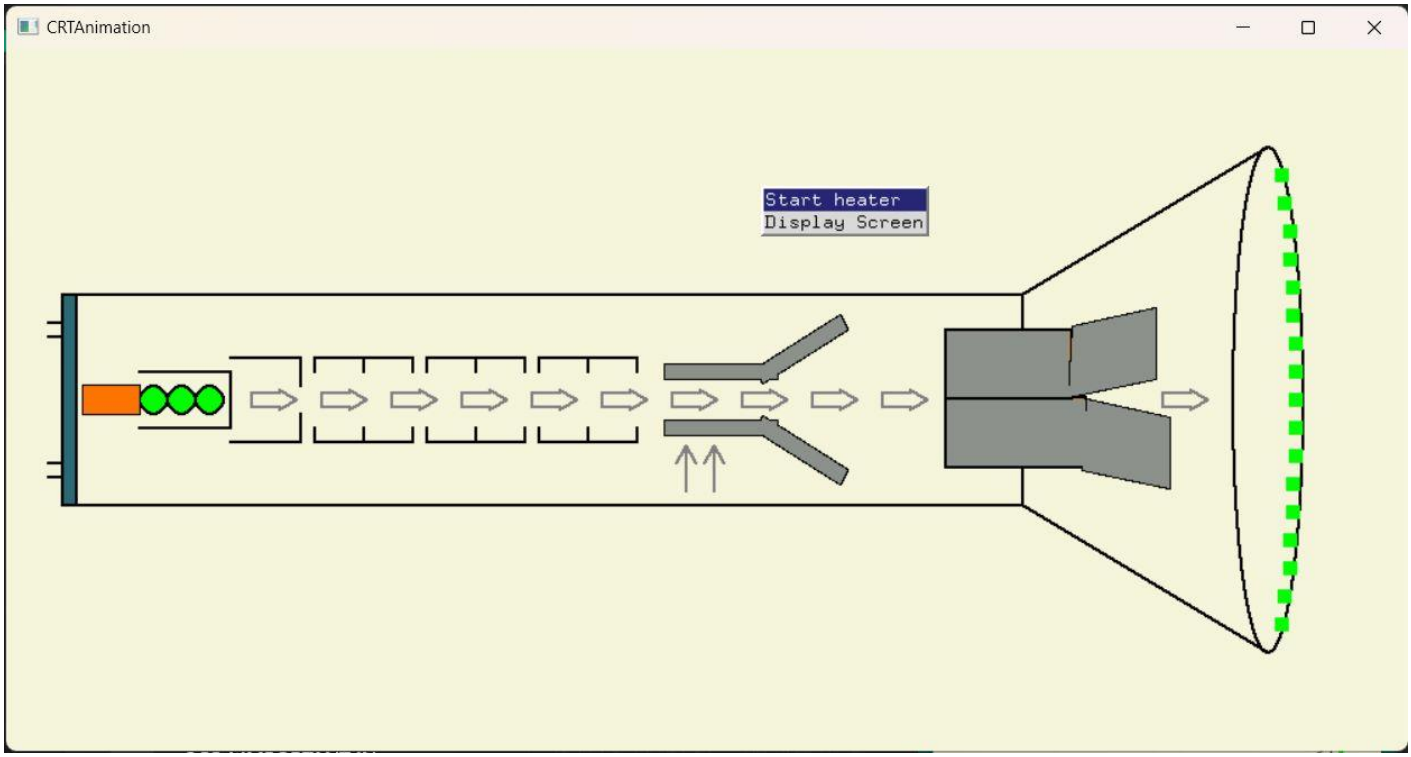
```
        glutAddMenuEntry("Display Screen",2);
        break;
    case 2:
        displayWave(100,60,45);
        break;
    default:
        exit(0);
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 500);
    glutInitWindowPosition(20, 100);
    glutCreateWindow("CRTAnimation");
    glClearColor(br, bg, bb, 0.5);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 1000, 0, 500);
    glutCreateMenu(menu);
    glutAddMenuEntry("Start heater", 1);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

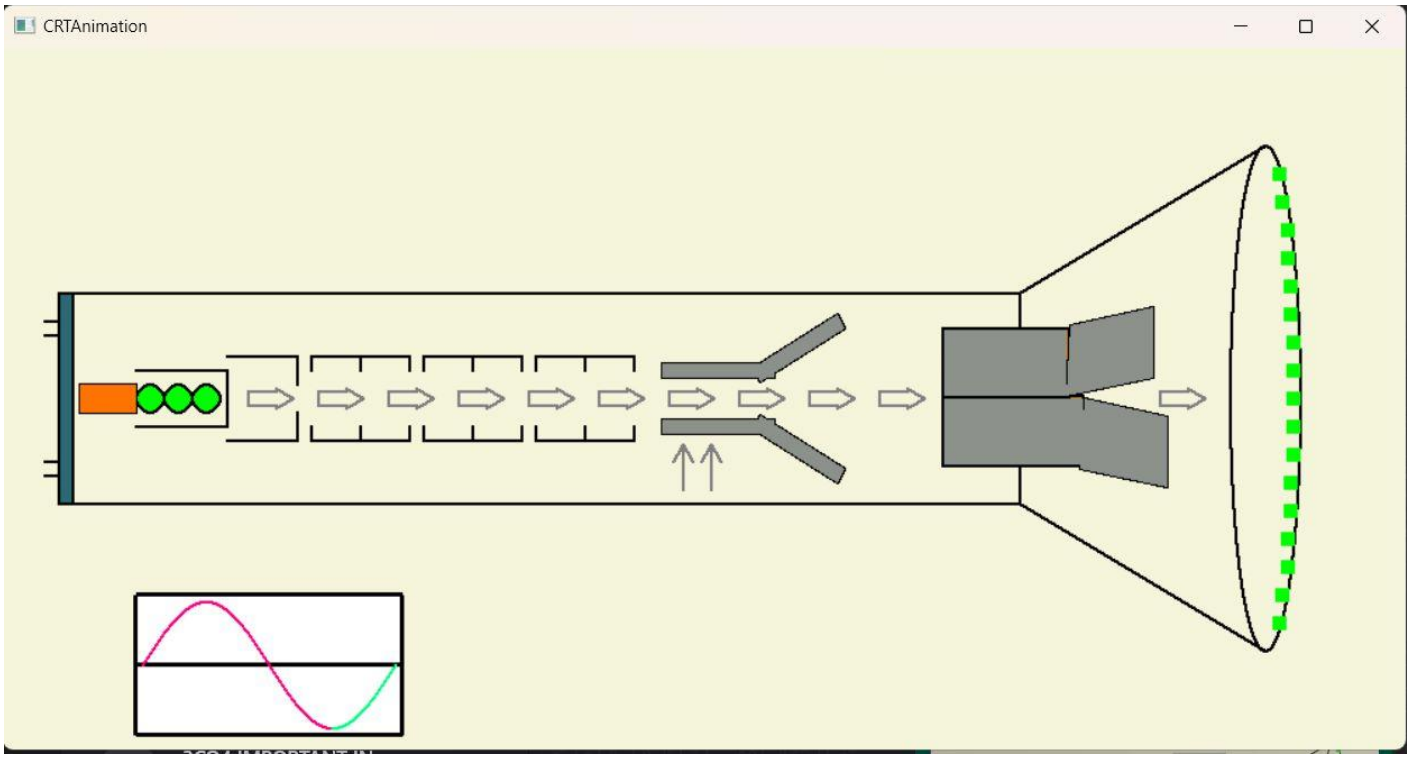
Output of the Code/Screenshots






CRTAnimation


CRTAnimation
Start heater
Display Screen


CRTAnimation