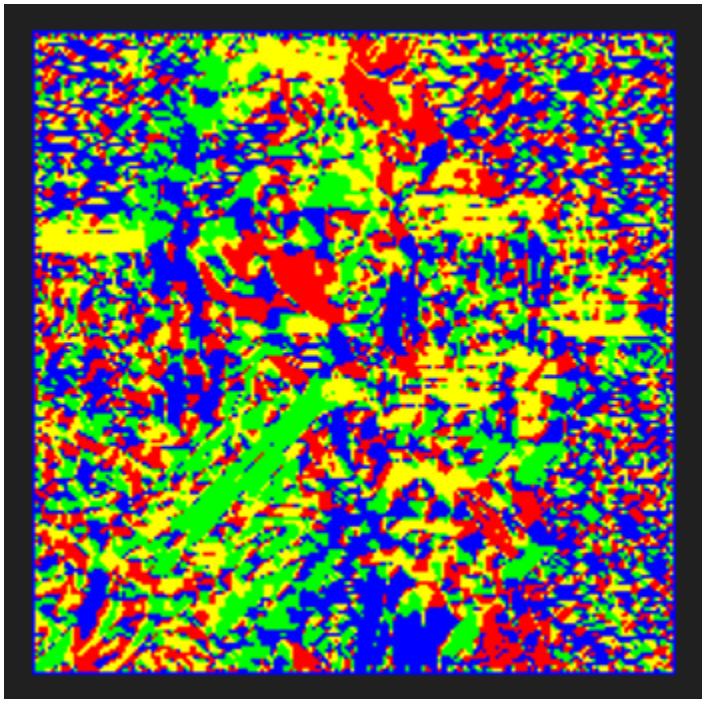
1. **Output Images**

OutputOrigin.bmp screenshot

OutputGauss.bmp screenshot

OutputGradient.bmp screenshot

OutputAngle.bmp screenshot



OutputNMS.bmp screenshot

OutputThres.bmp screenshot

2. Screenshots of simulation output in terminal

```
td=c++17
(systemc) bash-4.4$ ./sim.out

        SystemC 3.0.0-Accellera --- Aug 29 2024 18:54:01
        Copyright (c) 1996-2024 by all Contributors,
        ALL RIGHTS RESERVED
> SUCCESS: The file was read successfully.

 ----- BMP HEADER -----
> MAGIC NUMBER : BM
> FILE SIZE : 120056 bytes
> OFFSET OF BMP DATA : 0x36

 ----- DIB HEADER -----
 > NUMBER OF DIB HEADER : 0d40 bytes
 > WIDTH : 200 Pixels
 > HEIGHT : 200 Pixels
 > COLOR PLANE : 1 Plane
 > BITS/PIXEL : 24 bpp
 > COMPRESSION : 0
 > SIZE OF DATA : 120400 bytes
 > H-RESOLUTION : 2834 Pixels/Meter
 > V-RESOLUTION : 2834 Pixels/Meter
 > NUMBER OF PALETTE :  0
 > IMPORTANCE : 0

 ----- BMP DATA -----
 > Create memX[][] Array
 > Create memXG[][] Array
 > Create Gxy[][] Array
 > Create Theta[][] Array
 > Create bGxy[][] Array
 >> OUT: ORIGIN >>
 >> OUT: GAUSSIAN >>
 >> OUT: GRADIENT >>
 >> OUT: ANGLE >>
 >> OUT: NMS >>
 >> OUT: HYSTERESIS >>
 > 0th Matching Ratio : 100percent
 > 1th Matching Ratio : 100percent
 > 2th Matching Ratio : 99.865percent
 > 3th Matching Ratio : 100percent
 > 4th Matching Ratio : 98.4046percent
 > 5th Matching Ratio : 99.7968percent
```

3. Screenshots of code

Write_Data Function

```
void Canny_Edge::Write_Data() {
  if (!bCE.read() && !bWE.read()) {
    if(dWriteReg.read() == WRITE_REGX){
      regX[AddrRegRow.read()][AddrRegCol.read()] = InData.read();
      }
    // Insert Your Code here //
    // ...
    // ...
    // ...
    else if(dWriteReg.read() == WRITE_REGY)  //write regy option
      regY[AddrRegRow.read()][AddrRegCol.read()] = InData.read();
    else if(dWriteReg.read() == WRITE_REGZ)  //write regz option
      regZ[AddrRegRow.read()][AddrRegCol.read()] = InData.read();

    // For debug
    #if defined (_DEBUG_)
    cout << "@" << sc_time_stamp() << ":: Write: " << InData.read() << endl;
    #endif
  }
}
```

Read_Data Function

```
void Canny_Edge::Read_Data() {
  if (!bCE.read() && bWE.read()) {
    unsigned int dData;
    if(dReadReg.read() ==  REG_GAUSSIAN)    dData = Out_gf;
    // Insert Your Code here //
    // ...
    // ...
    // ...
    else if(dReadReg.read() == REG_GRADIENT){  //gradient operation
            dData = Out_gradient;
    }
    else if(dReadReg.read() == REG_DIRECTION) {  //direction operation
            dData = Out_direction;
    }
    else if(dReadReg.read() == REG_NMS){  //Non-maximum suppression operation
            dData = regX[AddrRegRow.read()][AddrRegCol.read()];
    }
    else if(dReadReg.read() == REG_HYSTERESIS){  //hysteresis operation
            dData = Out_bThres;
    }

    OutData.write(dData);

    // For debug
    #if defined (_DEBUG_)
    cout << "@" << sc_time_stamp() << ":: Read[" << AddrRegRow.read() << "]";
    cout << "[" << AddrRegCol.read() << "]: " << dData << endl;
    #endif
  }
}
```

MODE_SOBEL operation

```
  else if(OPMode.read() == MODE_SOBEL){
    int c,d;
    short Gx=0;    // X direction Component
    short Gy=0;    // Y direction Component

    // 1. input : Sobeldx, Sobeldy, regX(Gaussian Filtered Image)
    // 2. Output : Out_gradient(0~255), Out_direction(0, 45, 90, 135)
    // Insert Your Code here //
    // ...
    // ...
    // ...
     for(c=-1; c<=1; c++) {
              for(d=-1; d<=1; d++) {
                  Gx += regX[c+1][d+1] * Sobeldx[c+1][d+1];    //convolution with sobel operator
                  Gy += regX[c+1][d+1] * Sobeldy[c+1][d+1];
              }
          }

     Out_gradient = (abs(Gx) + abs(Gy)) / 2;       //alpha = 2
     if (Out_gradient > 255) {
      Out_gradient = 255;
     }


     //approximation to calculate direction
     //first step
     if(Gy<0){
       Gx = -1 * Gx;
       Gy = -1 * Gy;
     }
     else{
       Gx = Gx;
       Gy = Gy;
     }

     //second step
     if(Gx >= 0){
       if(Gy <= 0.4*Gx){
         Out_direction = 0;
       }
       else if(Gy > 0.4*Gx && Gy <= 2.4*Gx){
         Out_direction = 45;
       }
       else if(Gy > 2.4*Gx){
         Out_direction = 90;
       }
     }
     else{
       if(Gy <= -0.4*Gx){
         Out_direction = 0;
       }
       else if(Gy > -0.4*Gx && Gy <= -2.4*Gx){
         Out_direction = 135;
       }
       else if(Gy > -2.4*Gx){
         Out_direction = 90;
       }
     }


  }
```

MODE_NMS operation

```
    }
    else if(OPMode.read() == MODE_NMS){
      // 1. input : regX(Gradient Image), regY(Direction Image)
      // 2. Output : regX(Gradient Image)
      // Insert Your Code here //
      // ...
      // ...
      // ...

      for (int i = 1; i < 4; i++) {
        for (int j = 1; j < 4; j++) {
            // Only process valid gradient values
            if (regX[i][j] >= 0) {
                int C = regX[i][j]; // Current pixel value
                bool keepC = true; // Flag to determine if C should be kept

                // Determine neighbors based on direction
                if (regY[i][j] == 0) { // Horizontal
                    int A = regX[i][j - 1];
                    int B = regX[i][j + 1];
                    if (C >= A && C >= B) {
                        // Suppress neighbors A and B
                        regX[i][j - 1] = 0;
                        regX[i][j + 1] = 0;
                    } else {
                        // C is not a local maximum
                        keepC = false;
                    }
                } else if (regY[i][j] == 45) { // Diagonal
                    int A = regX[i - 1][j + 1];
                    int B = regX[i + 1][j - 1];
                    if (C >= A && C >= B) {
                        regX[i - 1][j + 1] = 0;
                        regX[i + 1][j - 1] = 0;
                    } else {
                        keepC = false;
                    }
                } else if (regY[i][j] == 90) { // Vertical
                    int A = regX[i - 1][j];
                    int B = regX[i + 1][j];
                    if (C >= A && C >= B) {
                        regX[i - 1][j] = 0;
                        regX[i + 1][j] = 0;
                    } else {
                        keepC = false;
                    }
                } else if (regY[i][j] == 135) { // Diagonal
                    int A = regX[i - 1][j - 1];
                    int B = regX[i + 1][j + 1];
                    if (C >= A && C >= B) {
                        regX[i - 1][j - 1] = 0;
                        regX[i + 1][j + 1] = 0;
                    } else {
                        keepC = false;
                    }
                }

                // Suppress pixel C if it's not a local maximum
                if (!keepC) {
                    regX[i][j] = 0;
                }
```

MODE_HYSTERESIS operation

```
    else if(OPMode.read() == MODE_HYSTERESIS){
      // You should use these two threshold values.
      unsigned short dThresHigh = 20;
      unsigned short dThresLow = 5;

      // 1. input : regX(Gradient Image), regY(Direction Image), regZ(On/Off Image)
      //            regZ[][]==1: On / regZ[][]==0: Off
      // 2. Output : Out_bThres (0(Off) or 1(On))
      // Insert Your Code here //
      // ...
      // ...
      // ...
      for (int i = 1; i < 4; i++) {
        for (int j = 1; j < 4; j++) {
        if (regX[i][j] >= dThresHigh) {
            regZ[i][j] = 1; // Strong pixel
        }
        else if (regX[i][j] <= dThresLow) {
            regZ[i][j] = 0; // Weak pixel, suppress
        }
        else { // Candidate pixel
            bool connectedToStrong = false;

            // Check neighbors based on direction in regY
            if (regY[i][j] == 0) { // Horizontal direction
                if (regX[i][j-1] >= dThresHigh || regX[i][j+1] >= dThresHigh) {
                    connectedToStrong = true;
                }
            } else if (regY[i][j] == 45) { // Diagonal
                if (regX[i-1][j+1] >= dThresHigh || regX[i+1][j-1] >= dThresHigh) {
                    connectedToStrong = true;
                }
            } else if (regY[i][j] == 90) { // Vertical direction
                if (regX[i-1][j] >= dThresHigh || regX[i+1][j] >= dThresHigh) {
                    connectedToStrong = true;
                }
            } else if (regY[i][j] == 135) { // Diagonal
                if (regX[i-1][j-1] >= dThresHigh || regX[i+1][j+1] >= dThresHigh) {
                    connectedToStrong = true;
                }
            }

            if (connectedToStrong) {
                regZ[i][j] = 1; // Keep candidate pixel
            } else {
                regZ[i][j] = 0; // Discard candidate pixel
            }
        }
        }
      }

      // Output for Hysteresis
      Out_bThres = regZ[AddrRegRow.read()][AddrRegCol.read()];

    }
  }
}
```

4. Some reasons for double edges are:

- The Gaussian filter smooths both noise and edges, leading to double edges.
- Improper threshold values can also lead to double edges.
- In the above-attached output images, double edges occur in textured regions because the intensity of pixels changes rapidly in both directions. As the gradient, NMS, and

hysteresis operations are performed based on the neighbors, these operations result in double edges on both sides of the textured edge.

- Some ways to prevent double edges include using a larger Gaussian kernel and using an adaptive threshold value for weak and strong pixels, which will account for the intensity of the local area instead of having a global value. Additionally, for textured regions, the gradient in only one direction should be considered.