محاضرة CS50 - التحدي في معالجة الصور

التحدي في معالجة الصور (Edge Detection)

في هذا التحدي، نتعامل مع عمليات معالجة الصور باستخدام تقنية **Sobel** لاكتشاف الحواف. يتم استخدام **Sobel filter** لتمييز الحواف في الصورة بناءً على التدرجات بين الألوان في الاتجاهين الأفقي والرأسي.

تتضمن التحديات التي واجهناها في هذا المشروع التعامل مع **المصفوفات الثنائية** وتطبيق **فلاتر Sobel** على صورة RGB. الفكرة الأساسية هي استخدام فلاتر **Sobel** في اتجاهات مختلفة لمعرفة التغيرات في الألوان بين البكسلات المجاورة لتحديد الحواف في الصورة.

شرح الدوال المستخدمة:

1. دالة تحويل الصورة إلى تدرج رمادي

```
void grayscale(int height, int width, RGBTRIPLE
image[height][width]) {
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int buf = round((image[i][j].rgbtBlue +
image[i][j].rgbtGreen + image[i][j].rgbtRed) / 3.0);
            image[i][j].rgbtBlue = buf;
            image[i][j].rgbtGreen = buf;
            image[i][j].rgbtRed = buf;
        }
    }
}
```

تقوم هذه الدالة بتحويل صورة ملونة إلى تدرج رمادي عبر حساب المتوسط الحسابي لكل من القيم الحمراء والخضراء والزرقاء للبكسل في الصورة. النتيجة هي صورة تحتوي فقط على تدرجات رمادية.

2. دالة عكس الصورة أفقياً:

```
void reflect(int height, int width, RGBTRIPLE image[height][width])
{
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width / 2; j++) {
            RGBTRIPLE buf = image[i][j];
            image[i][j] = image[i][width - j - 1];
            image[i][width - j - 1] = buf;
        }
    }
{
```

تقوم هذه الدالة بعكس الصورة أفقياً عبر تبديل البكسلات بين الجهة اليمنى واليسرى للصورة.

3. دالة تمويه الصورة (Blur):

```
void blur(int height, int width, RGBTRIPLE image[height][width]) {
    RGBTRIPLE copy[height][width];
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            copy[i][j] = image[i][j];
        }
    }

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int r = 0, g = 0, b = 0, count = 0;

            for (int di = -1; di <= 1; di++) {
                for (int dj = -1; dj <= 1; dj++) {
                    int ni = i + di, nj = j + dj;
                    if (ni >= 0 && ni < height && nj >= 0 && nj <
width) {
                        r += copy[ni][nj].rgbtRed;
                        g += copy[ni][nj].rgbtGreen;
                        b += copy[ni][nj].rgbtBlue;
                        count++;
                    }
                }
            }

            image[i][j].rgbtRed = round(r / (float)count);
            image[i][j].rgbtGreen = round(g / (float)count);
            image[i][j].rgbtBlue = round(b / (float)count);
        }
    }
{
```

تقوم هذه الدالة بتمويه الصورة عن طريق حساب المتوسط لكل لون (أحمر، أخضر، أزرق) للبكسلات المحيطة بالبكسل الحالي.

4. دالة اكتشاف الحواف باستخدام فلتر Sobel:

```
void edges(int height, int width, RGBTRIPLE image[height][width]) {
    RGBTRIPLE copy[height][width];
```

```
    int gx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
    int gy[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            copy[i][j] = image[i][j];
        }
    }

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (i == 0 || j == 0 || i == height - 1 || j == width -
1) {
                image[i][j] = copy[i][j];
                continue;
            }

            int rgx = 0, ggx = 0, bgx = 0;
            int rgy = 0, ggy = 0, bgy = 0;

            for (int di = -1; di <= 1; di++) {
                for (int dj = -1; dj <= 1; dj++) {
                    int ni = i + di, nj = j + dj;

                    if (ni >= 0 && ni < height && nj >= 0 && nj <
width) {
                        rgx += copy[ni][nj].rgbtRed * gx[di + 1][dj
+ 1];
                        rgy += copy[ni][nj].rgbtRed * gy[di + 1][dj
+ 1];
                        ggx += copy[ni][nj].rgbtGreen * gx[di +
1][dj + 1];
                        ggy += copy[ni][nj].rgbtGreen * gy[di +
1][dj + 1];
                        bgx += copy[ni][nj].rgbtBlue * gx[di + 1][dj
+ 1];
                        bgy += copy[ni][nj].rgbtBlue * gy[di + 1][dj
+ 1];
                    }
                }
            }
```

```
            int G_red = round(sqrt((rgx * rgx) + (rgy * rgy)));
            int G_green = round(sqrt((ggx * ggx) + (ggy * ggy)));
            int G_blu = round(sqrt((bgx * bgx) + (bgy * bgy)));

            G_red = (G_red > 255) ? 255 : G_red;
            G_green = (G_green > 255) ? 255 : G_green;
            G_blu = (G_blu > 255) ? 255 : G_blu;

            image[i][j].rgbtRed = G_red;
            image[i][j].rgbtGreen = G_green;
            image[i][j].rgbtBlue = G_blu;
        }
    }
}
```

تقوم هذه الدالة بتطبيق فلاتر **Sobel** لاكتشاف الحواف في الصورة بناءً على القيم التدرجية للألوان. يتم حساب قيم التدرج في الاتجاهين الأفقي والرأسي، ثم يتم حساب **الجذر التربيعي** لمجموع التدرجات في الاتجاهين للحصول على قيمة **التدرج النهائي** للبكسل.

دالة اكتشاف الحواف باستخدام فلتر Sobel
تقوم هذه الدالة بتطبيق فلاتر Sobel لاكتشاف الحواف في الصورة باستخدام طريقة التدرجات. يتم ذلك من خلال حساب التدرج الأفقي و التدرج الرأسي لكل بكسل في الصورة، ثم دمج التدرجين للحصول على التدرج النهائي الذي يمثل الحافة.

خطوات الدالة:
نسخ الصورة الأصلية:

أولاً، يتم نسخ الصورة الأصلية إلى مصفوفة copy لحفظ القيم الأصلية لكل بكسل. سيتم تعديل الصورة الأصلية بناءً على القيم المحسوبة.

```
RGBTRIPLE copy[height][width];
for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        copy[i][j] = image[i][j];
    }
}
```

تعريف فلاتر Sobel:

يتم تعريف مصفوفات الفلاتر الخاصة بـ Sobel في الاتجاهين الأفقي والرأسي:
فلتر Gx يستخدم للكشف عن التغيرات في الاتجاه الأفقي.
فلتر Gy يستخدم للكشف عن التغيرات في الاتجاه الرأسي.

```
int gx[3][3] = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
int gy[3][3] = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};
```

التعامل مع الحواف:

يتم التحقق من البكسلات التي تقع على الحواف (أي البكسلات التي تقع على الحدود العليا أو السفلى أو اليمنى أو اليسرى للصورة).
لا يتم تطبيق فلاتر Sobel على هذه البكسلات لأنها لا تحتوي على بكسلات محيطة كاملة، بل يتم نسخ قيمها كما هي من الصورة الأصلية.

```
if (i == 0 || j == 0 || i == height - 1 || j == width - 1) {
    image[i][j] = copy[i][j];
    continue;
}
```

حساب التدرجات Gx و Gy:

لكل بكسل في الصورة (باستثناء الحواف)، يتم حساب التدرج الأفقي (Gx) والتدرج الرأسي (Gy) عبر جمع قيم Sobel للألوان (أحمر، أخضر، أزرق) للبكسلات المحيطة.
يتم ضرب القيم المحيطة لكل بكسل في مصفوفات Sobel لكل من Gx و Gy.

```
int rgx = 0, ggx = 0, bgx = 0;
int rgy = 0, ggy = 0, bgy = 0;

for (int di = -1; di <= 1; di++) {
    for (int dj = -1; dj <= 1; dj++) {
        int ni = i + di, nj = j + dj;
        if (ni >= 0 && ni < height && nj >= 0 && nj < width) {
            rgx += copy[ni][nj].rgbtRed * gx[di + 1][dj + 1];
            rgy += copy[ni][nj].rgbtRed * gy[di + 1][dj + 1];
            ggx += copy[ni][nj].rgbtGreen * gx[di + 1][dj + 1];
            ggy += copy[ni][nj].rgbtGreen * gy[di + 1][dj + 1];
            bgx += copy[ni][nj].rgbtBlue * gx[di + 1][dj + 1];
            bgy += copy[ni][nj].rgbtBlue * gy[di + 1][dj + 1];
        }
    }
}
```

حساب التدرج النهائي:

بعد حساب التدرجات Gx و Gy للألوان (أحمر، أخضر، أزرق)، يتم دمج التدرجات باستخدام الجذر التربيعي لمجموع التدرجات المربعة في الاتجاهين الأفقي والرأسي.
يتم تقليص القيم الناتجة بحيث لا تتجاوز 255 (القيمة القصوى الممكنة للون في صورة RGB).

```
int G_red = round(sqrt((rgx * rgx) + (rgy * rgy)));
int G_green = round(sqrt((ggx * ggx) + (ggy * ggy)));
int G_blu = round(sqrt((bgx * bgx) + (bgy * bgy)));
```

```
G_red = (G_red > 255) ? 255 : G_red;
G_green = (G_green > 255) ? 255 : G_green;
G_blu = (G_blu > 255) ? 255 : G_blu;
```

تحديث قيم البكسل:

بعد حساب التدرجات، يتم تحديث قيم اللون للبكسل في الصورة الأصلية باستخدام القيم المحسوبة للتدرج النهائي.

```
image[i][j].rgbtRed = G_red;
image[i][j].rgbtGreen = G_green;
image[i][j].rgbtBlue = G_blu;
```

ملخص الاخطاء

1. مشكلة مع حواف الصورة

- **الخطأ**: تم تطبيق فلاتر Sobel على البكسلات الموجودة على الحواف (البكسلات في أطراف الصورة)، وهو أمر غير صحيح لأن الحواف ليس لها بكسلات محيطة بشكل كامل.
- **التصحيح**: يجب استبعاد البكسلات الواقعة على الحواف من معالجة الفلاتر. يمكن ذلك عبر التحقق من أن البكسل ليس في بداية أو نهاية الصفوف أو الأعمدة قبل تطبيق الفلاتر.

حل المشكلة في الكود:

```
if (i == 0 || j == 0 || i == height - 1 || j == width - 1) {
    image[i][j] = copy[i][j];
    continue;
}
```

2. عدم التعامل مع الصورة كـ "مصفوفة" من البكسلات بشكل صحيح

- **الخطأ**: عند حساب التدرجات باستخدام فلاتر **Sobel** (التدرج الأفقي Gx والتدرج الرأسي Gy)، قد يتم تجاوز الحدود المسموح بها لمؤشرات البكسلات.
- **التصحيح**: يجب التحقق من أن جميع المؤشرات التي تم الوصول إليها في الصورة (مثل `ni` و `nj` في الحلقة) تقع ضمن حدود الصورة.

حل المشكلة في الكود:

```
if (ni >= 0 && ni < height && nj >= 0 && nj < width)
```

3. عدم تطبيق القيم المناسبة عند حساب التدرجات

- **الخطأ**: قد تحدث أخطاء في الحسابات بسبب عدم ضرب القيم بأرقام الفلاتر بشكل صحيح.
- **التصحيح**: تأكد من ضرب القيم باستخدام الفلاتر **Sobel** للألوان الأحمر والأخضر والأزرق كما هو موضح في الكود.

حل المشكلة في الكود:

```
rgx += copy[ni][nj].rgbtRed * gx[di + 1][dj + 1];
rgy += copy[ni][nj].rgbtRed * gy[di + 1][dj + 1];
```

4. عدم معالجة القيم الناتجة بشكل صحيح

- **الخطأ**: عند حساب التدرج النهائي، قد تحصل على قيم خارج النطاق المسموح به (أي أكبر من 255 أو أقل من 0).
- **التصحيح**: يجب تقليص القيم الناتجة من الحسابات بحيث تبقى ضمن الحدود المسموح بها في نموذج **RGB** (أي بين 0 و 255).

حل المشكلة في الكود:

```
G_red = (G_red > 255) ? 255 : G_red;
G_green = (G_green > 255) ? 255 : G_green;
G_blu = (G_blu > 255) ? 255 : G_blu;
```

5. استخدام نسخ غير دقيقة لبيانات الصورة

- **الخطأ**: عند نسخ الصورة الأصلية إلى **copy**، إذا كانت هناك أي أخطاء في الطريقة التي يتم بها النسخ، قد يحدث تعديل غير صحيح على الصورة أثناء تنفيذ الفلاتر.
- **التصحيح**: تأكد من أنك تستخدم عملية نسخ دقيقة كما هو موضح في الكود، حيث يجب نسخ كل بكسل على حدة.

حل المشكلة في الكود:

```
copy[i][j] = image[i][j];
```

6. مشكلة في تنسيق الصورة الناتجة

- **الخطأ**: أحياناً يحدث خلط في تنسيق الصورة الناتجة بسبب الطريقة التي يتم بها تحديث قيم البكسلات، خاصة في حالة وجود أخطاء في الحسابات أو معالجة القيم بشكل غير صحيح.
- **التصحيح**: تأكد من تحديث الصورة بالطريقة الصحيحة باستخدام القيم المحسوبة بعد معالجة التدرجات.

حل المشكلة في الكود:

```
image[i][j].rgbtRed = G_red;
image[i][j].rgbtGreen = G_green;
image[i][j].rgbtBlue = G_blu;
```

7. عدم اختبار الكود بشكل كافٍ

- **الخطأ**: عدم اختبار الكود بشكل كافٍ على صور متنوعة قد يؤدي إلى اكتشاف أخطاء غير متوقعة عند العمل مع صور أكثر تعقيدًا.
- **التصحيح**: يجب إجراء اختبارات متعددة على الصور ذات الأحجام المختلفة والألوان المتنوعة للتأكد من صحة التطبيق. يمكن استخدام صور بسيطة (مثل 3x3 أو 4x4) لاختبار الفلاتر بشكل أولي.