# Finger Exercises 3

*Color, Alpha & Fills*

≈ 60 minutes, plus a 10-minute bonus

CS 453 — Computer Graphics

**What this is:** Short drills that lock in Lecture 3 — blending colors, drawing and filling circles, and flood fill. Nothing needs more than 5 minutes, except the starred bonus at the end.

**How to do it:** On **paper** first, no computer. If an item takes longer than 5 minutes, skip it, finish the rest, come back. Afterwards, verifying your answers in Python is excellent practice — and A1's `painter.py` is the natural place to try them.

---

## Part A • Do You Remember?

RECALL ≈ 5 min

**A1.** Alpha blending: with $\alpha$ the opacity of the new color $F$ over what is already in the pixel $B$, the result is $C =$ _____. With $\alpha = 0$ you see _____; with $\alpha = 1$ you see _____.

**A2.** Rearranged, the blend is $B + \alpha\,(F - B)$ — which is _____ with $t = \alpha$, the same formula that walked the float line in Lecture 2.

**A3.** Walking $x$ and computing $y = \sqrt{r^2 - x^2}$ leaves gaps at the _____ of the circle, because there the circle is _____ and drops several rows per column.

**A4.** The inside test for a filled disc: paint $(x, y)$ iff _____. We never take a square root because _____.

**A5.** Flood fill: take a pixel off the to-do list. If it is _____ or _____, skip it. Otherwise paint it and put its _____ neighbors on the list.

## Part B • Blend by Hand

COMPUTE ≈ 6 min

Channels are integers 0–255. Blend each channel separately, then round to nearest with `int(v + 0.5)`.

**B1.** Pure red $(255, 0, 0)$ at $\alpha = 0.4$ over mid-gray $(100, 100, 100)$. Result: ( _____, _____, _____ )

**B2.** Use the lerp form. Foreground channel 220, background channel 20, $\alpha = 0.25$: $20 + 0.25 \cdot$ ( _____ ) = _____

**B3.** White at $\alpha = 0.3$ over black: the exact value is _____. `int()` alone stores _____; `int(v + 0.5)` stores _____.

**B4.** Two 50% white discs stacked over black. After the first the pixel holds _____; after the second it holds _____. Why is the answer not 255?

## Part C • Chop or Round?

COMPUTE ≈ 5 min

Program A chops: `C = int(x)`. Program B rounds: `C = int(x + 0.5)`. Stack 20% white discs over a black pixel and fill the table. Each row uses the value *stored* by the row above.

| discs | Program A (chop) | Program B (round) | gap |
|---|---|---|---|
| 1 | 51 | 51 | 0 |
| 2 | _____ | _____ | _____ |
| 3 | _____ | _____ | _____ |
| 4 | _____ | _____ | _____ |

**C1.** The checker allows each channel to differ by at most 2. After which disc does Program A first sit *on* that limit? Which program is guaranteed a gap of 0, and why?

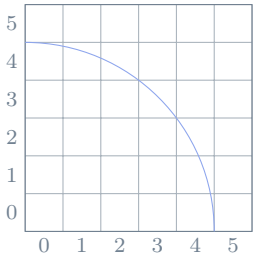
## Part D • Circles: Walk $x$, Then Mirror

TRACE ≈ 7 min

Attempt 1 from the lecture, for one quarter of a circle of radius 5 centered at the origin: for $x = 0, 1, \ldots, 5$ compute $y =$ `int(sqrt(25 - x*x) + 0.5)`.

**D1.** Fill the row of $y$ values, then shade the six pixels on the grid.

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $y$ | 5 | ____ | ____ | ____ | ____ | ____ |




**D2.** Between which two consecutive columns does $y$ jump by more than one row? How many rows are skipped there?

**D3.** Attempt 2 walks only while $x \le y$ (the first octant) and mirrors each point eight ways. For $r = 5$ that octant is the points _____. Mirroring gives $8 \times 4 = 32$ pixels — but how many are *distinct*, and why fewer?

**D4.** Mirror the octant point $(2, 5)$ around center $(10, 10)$. List all eight pixels.

## Part E • The Inside Test

COMPUTE ≈ 6 min

`fill_circle(img, 6, 6, 4, color)` — center $(6, 6)$, radius 4, so the test is $(x-6)^2 + (y-6)^2 \le 16$.

**E1.** For each pixel, write $d^2$ and tick in or out.

| pixel | $d^2$ | in | out |
|---|---|---|---|
| $(3, 3)$ | _____ | ☐ | ☐ |
| $(2, 6)$ | _____ | ☐ | ☐ |
| $(9, 8)$ | _____ | ☐ | ☐ |
| $(6, 10)$ | _____ | ☐ | ☐ |
| $(3, 9)$ | _____ | ☐ | ☐ |

**E2.** If the test used `<` instead of `<=`, which of the five pixels would change verdict?

**E3.** The loop tests only the box $x \in [c_x - r,\ c_x + r]$, $y \in [c_y - r,\ c_y + r]$. For this call that is _____ tests. For a radius-50 disc on an $800 \times 600$ image: _____ tests instead of _____.

**E4.** A disc centered at $(2, 2)$ with radius 4 hangs off the top-left corner. What does the loop do about the pixels with negative coordinates?

## Part F • Find the Bug

FIX ≈ 8 min

Each snippet draws *something* — just not the right thing. Name the bug and give the one-line fix.

**F1.** Symptom: the moon looks fine alone, but the moon glow (four stacked 15% discs) comes out a shade darker than the reference, and the mist test fails by 2–3 per channel.

```
def blend_channel(f, b, alpha):
    return int(alpha * f + (1 - alpha) * b)
```

Bug & fix: _____

**F2.** Symptom: filled discs are missing a single pixel at the top, bottom, left and right.

```
for y in range(cy - r, cy + r + 1):
    for x in range(cx - r, cx + r + 1):
        if (x - cx)**2 + (y - cy)**2 < r * r:
            set_pixel(img, x, y, color)
```

Bug & fix: _____

**F3.** Symptom: flood fill usually works, but on a region touching the left edge the paint reappears on the *right* edge; on a region touching the right edge it crashes with `IndexError`.

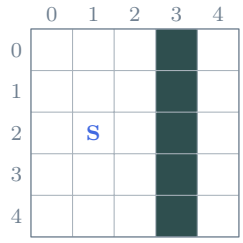
```
while todo:
    cx, cy = todo.pop()
    if img[cy][cx] != old:
        continue
    img[cy][cx] = new
    todo.append((cx + 1, cy))
    todo.append((cx - 1, cy))
    todo.append((cx, cy + 1))
    todo.append((cx, cy - 1))
```

Bug & fix: _____

## Part G • Trace the To-Do List

TRACE ≈ 8 min

A $5 \times 5$ image; column 3 is a dark outline, everything else is background. Call `flood_fill(img, 1, 2, blue)`. Use the lecture's version: `todo.pop()` takes the *last* item added; neighbors are added in the order right, left, down, up.




**G1.** Start: `todo = [(1,2)]`. Give the list after the first pop is fully processed, and after the second.

after pop 1: _____

after pop 2: _____

**G2.** Which is the first pop that gets *skipped*, and by which rule? Which is the second, and by which rule?

**G3.** When the list finally empties, how many pixels are blue? Is column 4 painted? Why or why not?

## Part H • Design on Paper

DESIGN ≈ 6 min

No code needed — a formula or a sentence each.

**H1.** A *ring*: outer radius $R$, inner radius $r$, centered at $(c_x, c_y)$. Write the inside test for one pass over the bounding box. Why is "fill the big disc, then fill the small one with the background color" a worse plan?

**H2.** An axis-aligned ellipse with half-widths $a$ (in $x$) and $b$ (in $y$) satisfies $\frac{(x-c_x)^2}{a^2} + \frac{(y-c_y)^2}{b^2} \le 1$. Rewrite it so the loop uses integers only.

**H3.** Change the flood fill to 8 neighbors. Which lines change? Then: the exemplar's mountains are outlined with Bresenham lines. Why would the 8-neighbor version paint the sky as well?

## Part I • One Sentence Each

EXPLAIN ≈ 5 min

**I1.** Why does the recursive flood fill survive a $20 \times 20$ doodle but die on a $300 \times 300$ region?

**I2.** Filling a disc by drawing outlines of radius $r, r-1, \ldots, 0$ leaves holes. Why?

**I3.** Stacked 15% discs brighten quickly at first, then more slowly. Explain with the lerp form.

**I4.** The checker tolerates $\pm 2$ per channel. Why does that not rescue a `blend_pixel` that truncates?

## Part J • Bonus: How Far Can the Drift Go? ★

COMPUTE ≈ 10 min

For the brave. Stack discs of opacity $\alpha$ forever. Let $g_n$ be the gap after $n$ discs between Program B (rounds) and Program A (chops), as in Part C. Program B is never wrong by more than half a step, and Program A never by more than a whole step.

**J1.** Explain why $g_{n+1} \leq (1-\alpha)\, g_n + 1$. (What does the next blend do to the old gap? What does one more chop add?)

**J2.** Show that if $g_n < 1/\alpha$ then $g_{n+1} < 1/\alpha$ too. Conclude that the gap is bounded for every $n$ — it cannot grow without limit.

**J3.** Evaluate the bound for $\alpha = 0.15$ and for $\alpha = 0.02$. The lecture measured a gap of 2 for $\alpha = 0.15$. Why is the real gap so far below the bound?

---

**Stuck on more than three items?** Re-read the Lecture 3 slides — every answer above comes straight from them.

# Answer Key

Check your work *after* attempting everything. If an answer surprises you, that item is worth re-reading in the slides.

**A1.** $C = \alpha F + (1-\alpha)B$; $\alpha = 0$ shows the **background** $B$ untouched; $\alpha = 1$ shows the **foreground** $F$ (fully opaque).

**A2.** **lerp** (linear interpolation): start at $B$, go fraction $\alpha$ of the way to $F$.

**A3.** At the **left and right sides**; there the circle is **steep** (nearly vertical) — Wednesday's naive line disease, again.

**A4.** $(x - c_x)^2 + (y - c_y)^2 \le r^2$; comparing squared distances gives the same verdict with integer arithmetic only — exact, and no float rounding.

**A5.** off the image; not the seed's original color (already painted, or the outline); its **4** neighbors — up, down, left, right.

**B1.** $(162, 60, 60)$. Red: $0.4 \cdot 255 + 0.6 \cdot 100 = 102 + 60 = 162$. Green and blue: $0 + 60 = 60$. A dull, pinkish red — 60% of the gray shows through.

**B2.** $20 + 0.25 \cdot 200 = 20 + 50 = 70$. One quarter of the way from 20 to 220.

**B3.** $0.3 \cdot 255 = 76.5$ exactly. `int(76.5)` $= 76$; `int(77.0)` $= 77$. An exact half is the case where the two disagree by a whole step.

**B4.** $127.5 \to 128$, then $0.5 \cdot 255 + 0.5 \cdot 128 = 191.5 \to 192$. Each disc adds half of the distance *still left* to white, never half of 255 — the build-up saturates and can never reach 255.

**C.** Disc 2: $51 + 0.8 \cdot 51 = 91.8$ — A stores 91, B stores 92, gap 1. Disc 3: A: $51 + 0.8 \cdot 91 = 123.8 \to 123$; B: $51 + 0.8 \cdot 92 = 124.6 \to 125$; gap 2. Disc 4: A: $51 + 0.8 \cdot 123 = 149.4 \to 149$; B: $51 + 0.8 \cdot 125 = 151.0 \to 151$; gap 2. A is always the darker one.

**C1.** After disc 3. Program B — the reference solution rounds the same way, so B reproduces it exactly; A's mistakes all push in one direction and never cancel.

**D1.** $y = 5, 5, 5, 4, 3, 0$. Pixels $(0,5)$, $(1,5)$, $(2,5)$, $(3,4)$, $(4,3)$, $(5,0)$ — the last one is all alone at the bottom of the $x = 5$ column.

**D2.** Between $x = 4$ ($y = 3$) and $x = 5$ ($y = 0$): rows 1 and 2 are skipped — two holes in the outline, exactly where the circle turns vertical.

**D3.** $(0,5)$, $(1,5)$, $(2,5)$, $(3,4)$ — the loop stops before $x = 4$ because $4 > 3$. Distinct pixels: 28. A point with $x = 0$ has only 4 distinct mirrors, not 8 ($-0 = 0$), so $(0,5)$ contributes 4 instead of 8.

**D4.** $(12, 15)$, $(8, 15)$, $(12, 5)$, $(8, 5)$ (flip signs of $x$ and $y$), then $(15, 12)$, $(5, 12)$, $(15, 8)$, $(5, 8)$ (swap $x$ and $y$ first, then flip signs).

**E1.** $(3,3)$: $9 + 9 = 18 > 16$, out. $(2,6)$: $16 + 0 = 16$, in — exactly on the rim, and `<=` keeps it. $(9,8)$: $9 + 4 = 13$, in. $(6,10)$: $0 + 16 = 16$, in (rim). $(3,9)$: $9 + 9 = 18$, out.

**E2.** $(2,6)$ and $(6,10)$ — both have $d^2 = 16$ exactly. Every pixel a whole radius away from the center in a straight line would drop out, and the checker's expected image keeps them.

**E3.** $9 \times 9 = 81$ tests. Radius 50: $101 \times 101 = 10{,}201$ tests instead of 480,000 — about 2% of the image, and the ratio only gets better for small discs.

**E4.** Nothing special — it tests them like any other and calls `set_pixel`, which silently drops anything off the image. Clipping comes for free from the contract set up in Lecture 1.

**F1.** `int()` truncates, so every layer rounds down and the drift accumulates in one direction. Fix: add `0.5` before converting — `int(alpha * f + (1 - alpha) * b + 0.5)`. Part C is this bug in numbers.

**F2.** `<` rejects pixels exactly on the rim, where $d^2 = r^2$ — the four pole pixels always, plus any other exact hits (like $(3,4)$ for $r = 5$). Fix: `<=`. One character; the checker notices.

**F3.** The bounds check is missing. Python treats `img[cy][-1]` as the *last* column (silent wrap-around to the right edge), while `img[cy][w]` raises `IndexError`. Fix: before the color check, `if not (0 <= cx < w and 0 <= cy < h): continue`.

**G1.** Pop 1 takes $(1,2)$, paints it, adds its neighbors: `[(2,2), (0,2), (1,3), (1,1)]`. Pop 2 takes the last one, $(1,1)$, paints it, adds four more: `[(2,2), (0,2), (1,3), (2,1), (0,1), (1,2), (1,0)]`. Note $(1,2)$ is back on the list — already painted.

**G2.** Pop 3 takes $(1,0)$ and paints it, adding $(2,0)$, $(0,0)$, $(1,1)$, $(1,-1)$. Pop 4 takes $(1,-1)$: skipped, **off the image**. Pop 5 takes $(1,1)$: skipped, **already painted** (no longer the old color).

**G3.** 15 — columns 0, 1, 2 in all five rows. Column 4 stays background: every path to it crosses the outline in column 3, and outline pixels are skipped, so nothing ever adds a column-4 pixel to the list. 4-connectivity cannot slip past a solid column.

**H1.** Paint $(x, y)$ iff $r^2 \le (x - c_x)^2 + (y - c_y)^2 \le R^2$ — one loop, one test with two comparisons. The two-disc plan destroys whatever was already under the hole (a gradient sky, a star), because "background color" is not one color.

**H2.** Multiply through by $a^2b^2$: paint iff $b^2(x-c_x)^2 + a^2(y-c_y)^2 \le a^2b^2$. No division, no square root — the same squared-land trick as the disc.

**H3.** Add four more `todo.append` lines for the diagonals $(c_x \pm 1, c_y \pm 1)$; nothing else moves. A Bresenham line steps diagonally on steep or shallow stretches, so two of its pixels touch only at a corner — an 8-connected fill squeezes through that corner and leaks out. 4-connectivity cannot, which is why the lecture chose it.

**I1.** Every unfinished pixel is one stack frame. A $20 \times 20$ region is at most 400 frames, under Python's $\sim$1000 limit; $300 \times 300$ is 90,000 frames, far over it — `RecursionError`.

**I2.** Integer-radius rings do not tile the grid: rounding puts the pixels of ring $k$ and ring $k+1$ on rows that skip some cells between them, and no ring ever claims those cells.

**I3.** Each disc adds $\alpha$ of the distance still left to white, $0.15 \cdot (255 - B)$. As $B$ climbs, that distance shrinks, so each step is smaller than the last — and the total can never pass 255.

**I4.** Tolerance absorbs *random* $\pm 1$ noise. Truncation is systematic: every layer pushes the same way, the gap reaches 2 by the third disc, and any further slip fails — and the anti-aliased lines of T8 blend through the same function.

**J1.** Both programs compute $\alpha F + (1-\alpha)B$ from *their own* stored $B$, so the old gap $g_n$ enters the new value scaled by $(1-\alpha)$. Then rounding versus chopping the same real number can differ by less than 1. Hence $g_{n+1} < (1-\alpha)g_n + 1$.

**J2.** $(1-\alpha)g_n + 1 < (1-\alpha)/\alpha + 1 = (1-\alpha+\alpha)/\alpha = 1/\alpha$. Since $g_1 \le 1 \le 1/\alpha$, induction gives $g_n < 1/\alpha$ for all $n$: the drift levels off instead of growing forever.

**J3.** $1/0.15 \approx 6.7$, so the gap never exceeds 6; $1/0.02 = 50$. The bound assumes chopping loses a *full* step every layer; on average it loses about half a step, and the chopped value sometimes lands on the same integer as the rounded one — so the real gap sits near $0.5/\alpha$ or below. Fainter discs are still far more dangerous: a quick Python check at 2% reaches a gap of 15.