Finger Exercises 5

Parametric Curves: Lines, Circles & Ellipses

≈ 60 minutes, plus a 10-minute bonus

CS 453 — Computer Graphics

What this is: Short drills that lock in Lecture 5 — describing a shape as a journey $\mathbf{p}(t)$, sampling it into points, circles and arcs, the ellipse's honest subtlety, rounding once at the end, and giving a line real width. Nothing needs more than 5 minutes, except the starred bonus at the end.

How to do it: On **paper** first, no computer. Sketch a little grid whenever a question has coordinates in it — the picture catches sign mistakes that the arithmetic hides. Angles are in radians unless a ° says otherwise. If an item takes longer than 5 minutes, skip it, finish the rest, come back. Friday adds the last tool — Bézier curves — on top of exactly these ideas.

Part A • Do You Remember?

RECALL ≈ 5 min

A1. $y = f(x)$ gives one y for each x, so it cannot draw a _____. A parametric curve answers a different question: not "how high above x?" but "_____".

A2. The straight line from A to B is $\mathbf{p}(t) =$ _____, with $t = 0$ at _____ and $t = 1$ at _____.

A3. In that formula $B - A$ is a _____, $t\,(B - A)$ is a _____, and $A +$ that is a _____ again.

A4. `sample_parametric(f, t0, t1, steps)` returns _____ points, because `steps` counts the _____, not the points.

A5. `circle_point`$(C, r, \theta) =$ (_____, _____). For an ellipse, the single r splits into _____ and _____.

A6. Curves live in _____; `bresenham_line` eats _____. Round _____ time(s), at the _____ moment.

A7. A thick line is not a line drawn several times — it is a _____, filled as _____.

Part B • The Dial by Hand

COMPUTE ≈ 6 min

Let $A = (1, 4)$ and $B = (9, 8)$, and $\mathbf{p}(t) = A + t\,(B - A)$.

B1. First write down the trip $B - A$. Then fill the table.

t	0	0.25	0.5	1
$\mathbf{p}(t)$	_____	_____	_____	_____

B2. Now compute $\mathbf{p}(2)$ and $\mathbf{p}(-0.5)$. The formula happily accepts them — what do they mean geometrically, and why does `sample_parametric` normally never ask for them?

B3. $A = (6, 2)$, $B = (6, 11)$ — a *vertical* line. Compute $\mathbf{p}(1/3)$. Why was this case awkward for $y = f(x)$ and completely ordinary here?

B4. In one line: why is "t is not x" worth saying out loud?

Part C • Fence Posts

COMPUTE ≈ 5 min

C1. `sample_parametric(f, 0, 1, steps=8)` returns _____ points. To get exactly 21 points you pass `steps` = _____.

C2. List every t value produced by `sample_parametric(f, 0, 1, steps=5)`.

C3. What does `steps=1` give? And `steps=0`?

C4. You sample a full circle with `steps=36` but return `steps` points instead of `steps + 1`. Describe exactly what the drawing looks like, and why.

C5. Rule of thumb from the slides: roughly how much curve length should one step cover, and roughly what `steps` covers most shapes on screen?

Part D • Circles by Hand

COMPUTE ≈ 7 min

A circle centred at $C = (6, 5)$ with radius $r = 3$.

D1. Compute `circle_point`$(C, 3, \theta)$ for $\theta = 0,\ \frac{\pi}{2},\ \pi,\ \frac{3\pi}{2}$.

D2. Compute the point at $\theta = \frac{\pi}{3}$. (Use $\cos\frac{\pi}{3} = 0.5$, $\sin\frac{\pi}{3} \approx 0.866$.)

D3. Our screen has y growing **down**. Take your four answers from C1 and say, *on screen*, which way the point travels as θ grows from 0.

D4. The slide built `circle_point` in three moves from $(\cos\theta, \sin\theta)$. Name them, in order.

Part E • Arcs Come Free

DESIGN ≈ 5 min

An arc is the same circle sampled over *part* of the dial. Remember y grows **down** on screen.

E1. You want the **lower** half of a circle on screen (a "smile"). Which range of θ?

E2. You want the **upper** half (a rainbow arch). Which range?

E3. A clock hand sweeps from 12 o'clock to 3 o'clock on screen. Give a θ range that draws that quarter arc, and say which θ is 12 o'clock.

E4. How much new machinery does an arc need beyond a full circle?

Part F • The Ellipse Dial Is Not the Rim Angle

COMPUTE ≈ 7 min

An ellipse centred at $C = (10, 6)$ with $r_x = 6$ and $r_y = 2$. Use $\cos 45° = \sin 45° \approx 0.707$, $\cos 60° = 0.5$, $\sin 60° \approx 0.866$.

F1. Compute `ellipse_point` at $\theta = 0°$, $90°$ and $180°$.

F2. Compute the point at $\theta = 45°$. Then measure the angle that point actually makes at the centre, using $\tan(\text{rim}) = (y - c_y)/(x - c_x)$.

F3. Same for $\theta = 60°$.

F4. From your two answers, write the general relationship between the dial θ and the rim angle, and say when the two coincide.

F5. So what *is* θ on an ellipse, in one sentence? And does any of this make the *drawing* wrong?

F6. A ball is driven around this ellipse by `ellipse_point`$(\theta = 2\pi t)$ with t ramping evenly from 0 to 1. Describe what a viewer sees, and where.

Part G • Where Rounding Hides

COMPUTE ≈ 5 min

G1. Fill the table. `int()` truncates *toward zero*; `round()` goes to the nearest whole number.

point	`int()`	`round()`
(7.6, 2.4)	_____	_____
(−3.2, 8.7)	_____	_____
(12.9, 0.3)	_____	_____

G2. For points with positive coordinates, which way does `int()` drag the whole curve on screen? Check your $(-3.2, 8.7)$ row — does the same description hold there?

G3. Why is a systematic half-pixel bias more visible on a shallow arc than on a steep one?

G4. State the rule that scales all the way to Week 14.

Part H • A Thick Line, Corner by Corner

COMPUTE ≈ 7 min

A stroke from $A = (1, 2)$ to $B = (7, 10)$ with width $w = 5$. Recall `perp`$(x, y) = (-y, x)$.

H1. Compute $B - A$ and its length. (The numbers are kind.)

H2. Compute the unit direction $\mathbf{d}$, then $\mathbf{n} = \texttt{perp}(\mathbf{d}) \cdot \frac{w}{2}$.

H3. Give the four corners of the rectangle.

H4. Check your work: verify $\mathbf{n}$ is perpendicular to $\mathbf{d}$ with one dot product.

H5. How is the rectangle actually filled, and which two guards must `thick_line` check before any of this arithmetic runs?

Part I • Find the Bug

FIX ≈ 8 min

Each snippet is *almost* right. Name the bug and give the fix.

I1. Symptom: every circle you draw has a small notch where the outline fails to close.

```
pts = []
for i in range(steps):
    t = t0 + (t1 - t0) * i / steps
    pts.append(f(t))
```

Bug & fix: _____

I2. Symptom: a shallow arc wobbles, and the whole curve sits a little up and to the left of where it should.

```
painter.bresenham_line(img, int(p.x), int(p.y),
                       int(q.x), int(q.y), color)
```

Bug & fix: _____

I3. Symptom: `ellipse_point` draws a perfect circle no matter what r_y you pass.

```
x = cx + rx * math.cos(theta)
y = cy + rx * math.sin(theta)
```

Bug & fix: _____

I4. Symptom: `thick_line` crashes with a division by zero, but only sometimes.

```
d = (b - a).normalized()
n = d.perp() * (width / 2)
```

Bug & fix: _____

I5. Symptom: asking for a 90° arc draws almost the whole circle several times over.

```
pts = sample_parametric(
    lambda t: circle_point(c, r, t), 0, 90, steps=40)
```

Bug & fix: _____

Part J • One Sentence Each

EXPLAIN ≈ 5 min

J1. Why can `sample_parametric` draw a circle, a spiral and a Lissajous figure without a single line of code changing?

J2. Give the one-line recipe that turns any $\mathbf{p}(t)$ into pixels, naming both functions.

J3. Animation reuses today's idea with nothing new. What plays the part of the dial, and what is its formula?

J4. Why are animation frames written as `frame_0007.ppm` rather than `frame_7.ppm`?

J5. Name each function's trap: `sample_parametric`, `draw_polyline`, `thick_line`, `ellipse_point`.

Part K • Bonus: Constant Speed on an Ellipse ★

DESIGN ≈ 10 min

For the brave. Part F showed that an evenly-turning dial does *not* move a ball evenly. Here is how animators fix it — with a table, not with calculus.

Take the quarter ellipse $r_x = 6$, $r_y = 2$ centred at the origin, and sample it at four equally spaced dial values $\theta = 0°, 30°, 60°, 90°$.

K1. Write down the four points. (Use $\cos 30^\circ \approx 0.866$, $\sin 30^\circ = 0.5$.)

K2. Compute the straight-line distance between each neighbouring pair. What do the three numbers tell you?

K3. Build the running total (the cumulative length from $\theta = 0^\circ$), and give the total length of the quarter.

K4. Now the real question: which dial value θ lands the ball **halfway along the rim**? Find the cumulative distance you are aiming for, locate it between two table entries, and interpolate *linearly between them* to estimate θ.

K5. Compare your answer with 45°, the halfway point of the *dial*. In one sentence, say what this procedure is doing and why the lecture called constant speed "not free".

Stuck on more than three items? Re-read the Lecture 5 slides — every answer above comes straight from them. The toolkit table near the end ("Today's Toolkit — and Each One's Trap") is the place to start.

Answer Key

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

A1. a circle (or any shape that doubles back — two y's for one x). The new question is "where am I at time t?" — describe the *journey*, not the height.

A2. $\mathbf{p}(t) = A + t\,(B - A)$; $t = 0$ gives A, $t = 1$ gives B.

A3. a **vector** (the trip); a **scaled trip**; a **point** (you arrive). Monday's meaning table, typed out.

A4. `steps + 1` points — `steps` counts the **gaps** (fence-posts), and both ends are included.

A5. $(c_x + r\cos\theta,\ c_y + r\sin\theta)$. For the ellipse: r_x and r_y — $(c_x + r_x\cos\theta,\ c_y + r_y\sin\theta)$.

A6. floats; ints. Round **once**, at the **last** moment, where the math meets pixels. Full precision everywhere before that.

A7. a **rectangle** (four corners stepped off the centreline), filled as **two triangles**.

B1. $B - A = (8, 4)$. Then $(1, 4)$, $(3, 5)$, $(5, 6)$, $(9, 8)$. Each quarter of t adds $(2, 1)$ — equal steps of the dial, equal steps along the line.

B2. $\mathbf{p}(2) = (17, 12)$ and $\mathbf{p}(-0.5) = (-3, 2)$. They lie on the infinite *line* through A and B, but outside the *segment* — beyond B and behind A. Nothing in the algebra stops you; we just normally sweep t from 0 to 1, which is exactly the segment.

B3. $B - A = (0, 9)$, so $\mathbf{p}(1/3) = (6, 2 + 3) = (6, 5)$. For $y = f(x)$ a vertical line has infinite slope — a special case needing its own code. Here t measures *progress along the journey*, not horizontal distance, so nothing is special: the x component simply never changes.

B4. t is progress along the path (0 start, 1 end) and knows nothing about horizontal. Reading t as x makes you expect vertical lines, loops and circles to break — they do not.

C1. 9 points; `steps` = 20. Always `steps + 1` points.

C2. $t = 0,\ 0.2,\ 0.4,\ 0.6,\ 0.8,\ 1.0$ — six values, five gaps of 0.2.

C3. `steps=1` gives 2 points — just the two ends, $[f(t_0), f(t_1)]$. `steps=0` is below the minimum: the lecture's convention is to return `[f(t0)]` alone, one point, rather than crash or divide by zero.

C4. The circle has a **notch** — a single missing 10° wedge of outline. Dropping the last point means the polyline stops one sample short of coming back to the start, so the ring never closes. (Drawing it `closed=True` would hide the symptom by joining last to first.)

C5. A step every 2–3 pixels of curve length; `steps` ≈ 64 covers most on-screen shapes. Too few and the polygon shows; too many is wasted work for the same picture.

D1. $(9, 5)$, $(6, 8)$, $(3, 5)$, $(6, 2)$ — right, then $+y$, then left, then $-y$. Each is the centre plus 3 in one axis direction.

D2. $(6 + 3(0.5),\ 5 + 3(0.866)) = (7.5,\ 7.598)$.

D3. $(9, 5)$ is right of centre; $(6, 8)$ has larger y, so it is *below* centre on screen; $(3, 5)$ is left; $(6, 2)$ is above. Right → down → left → up is **clockwise on screen**. Nothing to fix — the same formula that turns counter-clockwise on paper turns clockwise here, because y is flipped.

D4. (1) the unit picture $(\cos\theta, \sin\theta)$ — radius 1 at the origin; (2) **stretch** by r; (3) **move** to the centre by adding C.

E1. θ from 0 to π. There $\sin\theta \geq 0$, so $y = c_y + r\sin\theta \geq c_y$ — larger y is *lower* on screen.

E2. θ from π to 2π, where $\sin\theta \leq 0$ pulls y above the centre on screen.

E3. 12 o'clock is straight up on screen, which is $\theta = 3\pi/2$ (it makes $y = c_y - r$); 3 o'clock is $\theta = 0$, i.e. 2π. So sweep θ from $3\pi/2$ to 2π. (Any equivalent range, such as $-\pi/2$ to 0, is fine.)

E4. None. Same `circle_point`, same `sample_parametric` — only t_0 and t_1 change. Rainbows, smiles, pie slices and clock hands are all one sub-range of the dial.

F1. $(16, 6)$, $(10, 8)$, $(4, 6)$ — the ends of the two axes. At the axes the dial and the rim agree exactly.

F2. $(10 + 6(0.707),\ 6 + 2(0.707)) = (14.243,\ 7.414)$. Offsets from the centre: $(4.243,\ 1.414)$, so $\tan(\text{rim}) = 1.414/4.243 = 1/3$ and the rim angle is about 18.4° — *not* 45°.

F3. Offsets $(3,\ 1.732)$, point $(13,\ 7.732)$; $\tan(\text{rim}) = 1.732/3 = 0.577$, rim = 30° exactly. The dial ran 45° → 60° while the rim only crawled 18.4° → 30°.

F4. $\tan(\text{rim}) = \dfrac{r_y}{r_x}\tan\theta$. They coincide only when $r_x = r_y$ (a circle) — and, for any ellipse, at the axis points 0°, 90°, 180°, 270°, where both are the same by symmetry.

F5. θ is a **dial you turn** — a knob that sweeps out the rim — not an angle you can measure on the shape. The drawing is fine: the polyline still visits real points of the ellipse in order, so the shape is correct. Only the *spacing* of the samples is uneven.

F6. The ball does *not* move at constant speed. Equal steps of θ cover more rim near the ends of the long axis, so the ball visibly **speeds up** at the flat left and right ends and **crawls** over the top and bottom. "Constant speed" on an ellipse is not free.

G1. $(7, 2)$ and $(8, 2)$; $(-3, 8)$ and $(-3, 9)$; $(12, 0)$ and $(13, 0)$.

G2. For positive coordinates `int()` always shrinks both numbers, dragging the curve toward the **top-left**. But it truncates toward *zero*, not downward: `int`$(-3.2) = -3$ moves that x to the *right*. So the bias is not even a consistent direction — it is a fold toward the origin, which is worse than a uniform shift.

G3. A shallow arc changes y slowly, so many consecutive samples sit near the same pixel row; truncation pushes them onto a row and then abruptly off it, and the eye reads the resulting stair-steps as a **wobble**. On a steep arc the true y is changing fast enough that the same error is swallowed by the genuine slope.

G4. Keep **full float precision** through every computation; round **once**, at the last moment, where math meets pixels.

H1. $B - A = (6, 8)$, length $\sqrt{36 + 64} = \sqrt{100} = 10$ — a 6-8-10 triangle.

H2. $\mathbf{d} = (6, 8)/10 = (0.6,\ 0.8)$. `perp`$(\mathbf{d}) = (-0.8,\ 0.6)$, and $w/2 = 2.5$, so $\mathbf{n} = (-2,\ 1.5)$. Check: $|\mathbf{n}| = \sqrt{4 + 2.25} = 2.5$ — half the width, as it must be.

H3. $A + \mathbf{n} = (-1,\ 3.5)$; $B + \mathbf{n} = (5,\ 11.5)$; $B - \mathbf{n} = (9,\ 8.5)$; $A - \mathbf{n} = (3,\ 0.5)$. Step off both sides of both ends.

H4. $(0.6)(-2) + (0.8)(1.5) = -1.2 + 1.2 = 0$. A zero dot product means a right angle, so the rectangle really is square to the line.

H5. As **two triangles** (split along a diagonal). Guards: $w \le 0$, and $A = B$ — a zero-length line has no direction, so `normalized` would divide by zero and `perp` would be meaningless. Both cases draw **nothing**.

I1. The loop stops one sample short: `range(steps)` yields `steps` values, so the final $t = t_1$ is never evaluated and the ring never returns to its start. Fix: `range(steps + 1)` — `steps` counts gaps, and both ends are included.

I2. `int()` truncates instead of rounding, so every coordinate is biased toward zero rather than to the nearest pixel. Fix: use `round()` on all four values. Round once, here at the end — not earlier.

I3. The second line uses `rx` where it needs `ry`, so both axes get the same stretch. Fix: `y = cy + ry * math.sin(theta)`. An ellipse is one formula with *two* radii.

I4. No guard for $A = B$. A zero-length segment has no direction, so `normalized` divides by a length of zero. Fix: return early — draw nothing — when `width <= 0` or when A and B coincide (compare the squared length against a small `EPSILON`, not with `==`).

I5. `math.cos` and `math.sin` take **radians**, so 90 is not a quarter turn but about 14.3 full turns. Fix: sweep 0 to `math.pi / 2`, or convert with `math.radians(90)`.

J1. It never asks *what* `f` is — functions are values in Python, so it just calls the one it was handed. Every new curve is a new `lambda`, not a new sampler.

J2. `sample_parametric` to get a list of points, then `draw_polyline` to join them — together they are `draw_curve`.

J3. The **frame index** becomes the dial: $u = i/(\text{FRAMES} - 1)$, which runs 0 to 1 across the whole clip. The same $\mathbf{p}(t)$ that *drew* the path now *moves* something along it.

J4. Unpadded names sort as text: $1, 10, 11, 2, \ldots$ — so the frames are glued together in the wrong order and the movie is shuffled. Zero-padding to a fixed width makes alphabetical order match numerical order.

J5. `sample_parametric`: `steps + 1` points, both ends included. `draw_polyline`: `round()`, never `int()`. `thick_line`: $A = B$ has no `perp` — draw nothing. `ellipse_point`: θ is a dial, not the rim angle.

K1. (6, 0), (5.196, 1.0), (3.0, 1.732), (0, 2).

K2. 1.283, 2.315, 3.012. Three equal steps of the dial produce chords that differ by a factor of well over two — the ball would cover the last stretch nearly 2.3 times faster than the first. This *is* the speeding-up you predicted in Part F, now as numbers.

K3. 0, 1.283, 3.598, 6.610. The quarter arc is about 6.61 units long. (The true value is a little higher — chords cut the corner — but four samples already give a usable estimate, and more samples tighten it.)

K4. Half of 6.610 is 3.305. That falls between the entries for 30° (1.283) and 60° (3.598). The fraction of the way across is $(3.305 - 1.283)/(3.598 - 1.283) = 2.022/2.315 \approx 0.873$, so $\theta \approx 30 + 0.873 \times 30 \approx 56°$.

K5. The rim's midpoint sits at about 56°, a long way from the dial's midpoint of 45° — more evidence that θ is a knob, not a measurement. The procedure builds an **arc-length table** once, then looks up the dial value for each desired distance, so an evenly ticking clock produces evenly spaced positions. It is not free because it costs an extra pass of sampling, a stored table, and a search per frame — which is exactly why we skip it whenever we are only *drawing* the shape.