Lecture 1: What Graphics Is (and Isn't)

First Pixels & Gradients

CS 453 — Computer Graphics

Fakhir Shaheen

Forman Christian University

Course Information

Instructor

- Fakhir Shaheen
  - fakhirshaheen@fccollege.edu.pk
  - linkedin.com/in/fakhirshaheen
- Office:
  - S-426 (E)
  - Hours: TR 9:15 am - 11:00 am
    W 9:00 am - 9:30 am

Grading Breakdown

Component	**Weight**
Assignments *(weekly)*	40%
Project	20%
Midterm	15%
Final exam	25%
Total	**100%**

- 11 assignments, one per week
- Final project: **your** renderer, **your** scene
- No attendance policy

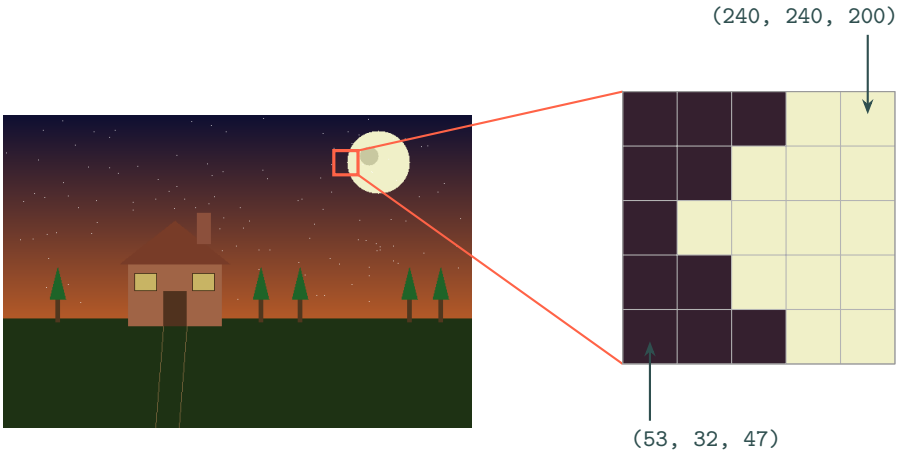
The Rules of the Game

- 3 lectures a week $\times$ 50 minutes — MWF
- An assignment **every week** (A1 drops **Friday**)
- Deadlines are **hard**: due Friday 11:59 pm, no late days
- All work is **individual**
- AI: may *explain* & *debug* — never *generate*. Disclose what you used.

What Graphics Is (and Isn't)

Every Image You Have Ever Seen . . .

. . . is a **grid of colored numbers**.



This course: learning to **compute** those numbers.

The One Idea of the Whole Semester

scene $\longrightarrow$ `<your code>` $\longrightarrow$ image

What This Course Is *Not*

Not: using the tools

- × Blender modeling
- × Photoshop / Illustrator
- × Unity / Unreal clicking
- × "prompt, then download"

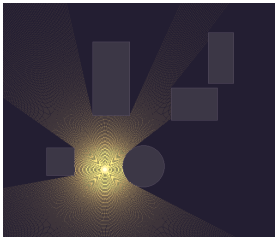
Instead: *building* the tools

- ✓ every pixel: **your** code
- ✓ a ray tracer, from scratch
- ✓ then the real GPU pipeline
- ✓ math only when a project needs it

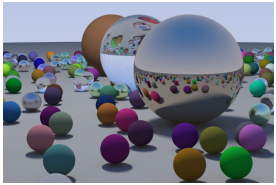
After this course, Blender & Unity stop being magic — you'll know what they do inside.

The Semester, in Pictures

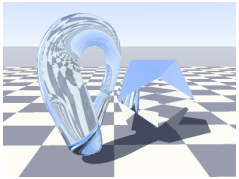
W1 — first pixels



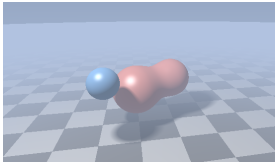
W4 — 2D light & shadow



W7 — ray tracer



W8 — real 3D models

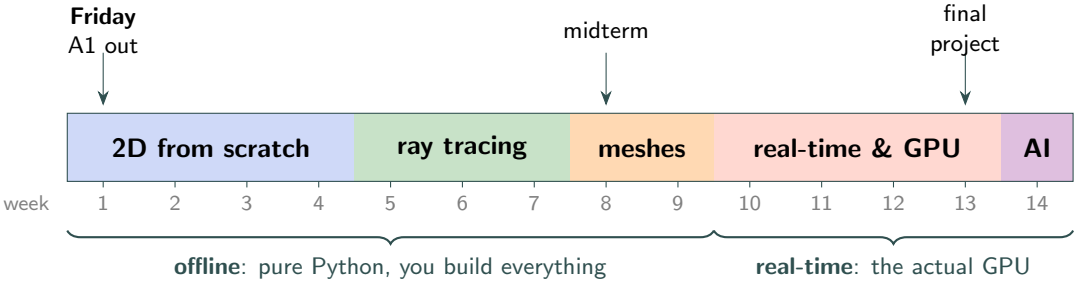


W11 — live GPU shaders

W14 — graphics meets AI

Every one rendered by code **you** will write — no libraries, no engines.

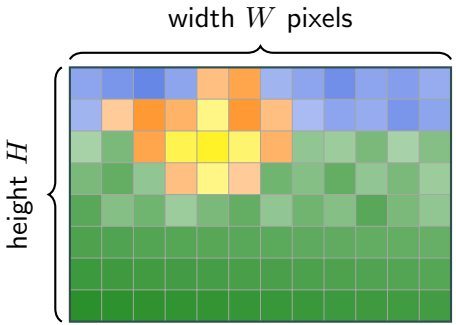
The Road Map



An assignment **every week**: A1 . . . A11, then the final project.

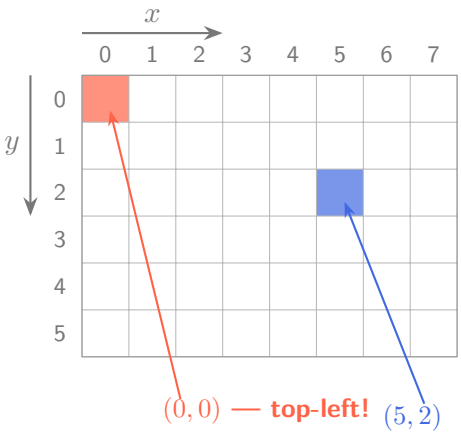
An Image Is a Grid of Numbers

Pixels



- **pixel** = *pic*ture *el*ement
- one pixel = one color
- resolution $W \times H$
- e.g. a 600×400 image
  $= 240{,}000$ pixels
  your phone: ~ 3 million

Which Pixel? — Coordinates



- x grows **rightward**
- y grows **downward**
- $(0, 0) =$ **top-left**

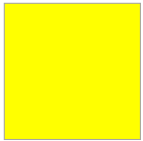
Flipped from math class!

Screens count rows the way you read: top to bottom.

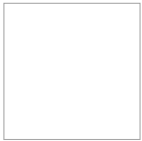
A Color Is Three Numbers

`(r, g, b)` — each channel $0 \ldots 255$

`(255, 0, 0)`



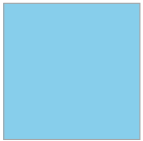
`(255, 255, 0)`



`(255, 255, 255)`

`(0, 0, 0)`

`(128, 128, 128)`

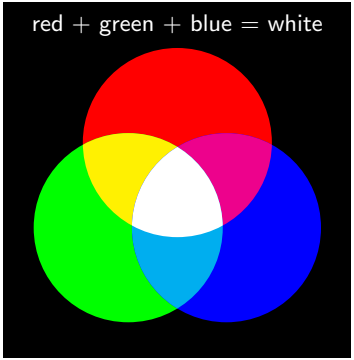


`(135, 206, 235)`

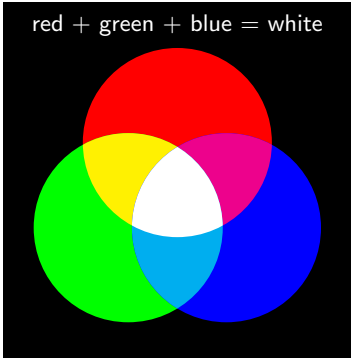
one byte per channel $\Rightarrow$ $256^3 \approx$ **16.7 million** colors

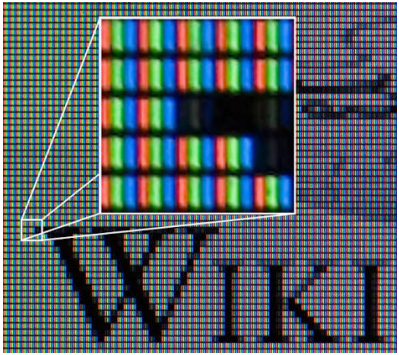
equal channels $\Rightarrow$ gray; all 0 $\Rightarrow$ black; all 255 $\Rightarrow$ white

Why Red, Green, Blue? Light *Adds*



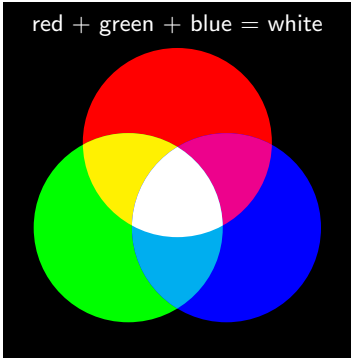
Why Red, Green, Blue? Light *Adds*

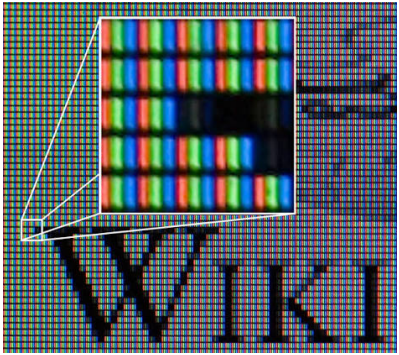




Your screen really is three tiny lights per pixel — look closely.

Why Red, Green, Blue? Light *Adds*





Your screen really is three tiny lights per pixel — look closely.

Screens **emit** light, so colors *add* — unlike paint, which absorbs.

First Pixels

Setup — There Is None

A1 uses **only the Python standard library**.

```
$ python3 --version
Python 3.12.4          # anything >= 3.9: done
```

Setup — There Is None

A1 uses **only the Python standard library**.

```
$ python3 --version
Python 3.12.4          # anything >= 3.9: done
```

- An image is a **block of numbers** written to disk — a few lines of plain Python.
- Bonus: it runs on **PyPy** too, whose JIT makes pixel loops far faster. Matters by Week 5.

Your Sketchbook: a `list` of Rows

An image *is* a list of rows

```
navy = (16, 18, 34)
img = [[navy] * 640 for _ in range(420)]  # 420 rows of 640

img[5][10] = (255, 0, 0)          # write:  row 5, column 10
print(img[5][10])                 # -> (255, 0, 0)
print(len(img), len(img[0]))      # -> 420 640  (rows, columns)
```

Your Sketchbook: a `list` of Rows

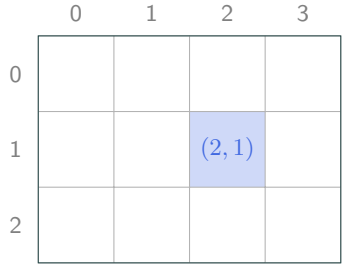
An image *is* a list of rows

```
navy = (16, 18, 34)
img = [[navy] * 640 for _ in range(420)]  # 420 rows of 640

img[5][10] = (255, 0, 0)          # write:  row 5, column 10
print(img[5][10])                 # -> (255, 0, 0)
print(len(img), len(img[0]))      # -> 420 640  (rows, columns)
```

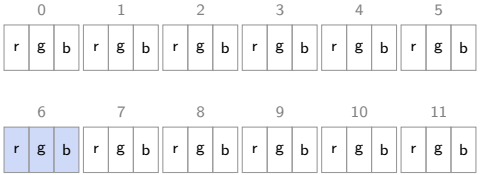
- Note the order: `img[y][x]` — **row first**, then column.
- Also note: `[[navy] * 640] * 420` makes 420 aliases of *one* row.
- Python won't stop you writing off-image (`img[-1]` is the *last* row!) — clipping is *your* job (A1, task T1).

Saving It: Rows Become One Flat Run of Bytes



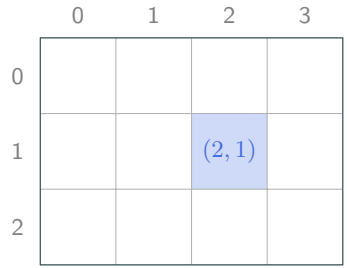
`img[y][x]` — 3 rows of 4 tuples

saving flattens



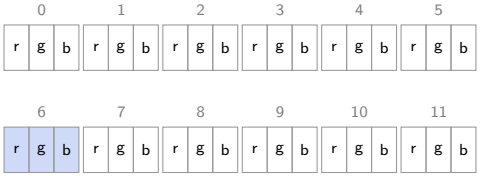
12 pixels = 36 bytes, row after row

Saving It: Rows Become One Flat Run of Bytes



`img[y][x]` — 3 rows of 4 tuples

saving flattens

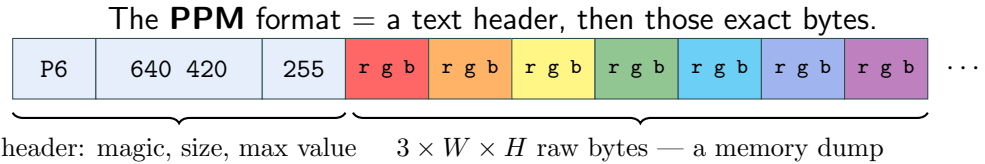


12 pixels = 36 bytes, row after row

row by row, top to bottom $\Rightarrow$ pixel (x, y) lands at byte $3\,(y \cdot W + x)$

. . . which is exactly why we index `img[y][x]`: row first.

An Image *File* Is the Same Thing



- An image file is **not** a mysterious object: header + memory dump.
- PNG / JPEG mostly just *compress* these same bytes.

The Whole "Format" in Nine Lines: save_ppm

```python
def save_ppm(img, path):
    W, H = len(img[0]), len(img)
    data = bytearray()
    for row in img:
        for (r, g, b) in row:
            data += bytes((r, g, b))
    with open(path, "wb") as f:
        header = f"P6\n{W} {H}\n255\n"
        f.write(header.encode("ascii"))
        f.write(data)
```

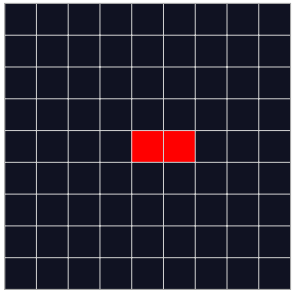
- the header, then the bytes
- open() and a loop
- in A1 this is GIVEN at the top of painter.py — the *only* function that touches a file. Read it; don't modify it.

Your First Pixels (For Real This Time)

```
navy = (16, 18, 34)
img = [[navy] * 64 for _ in range(64)]

img[32][32] = (255, 0, 0)
img[32][33] = (255, 0, 0)

save_ppm(img, "first.ppm")
```



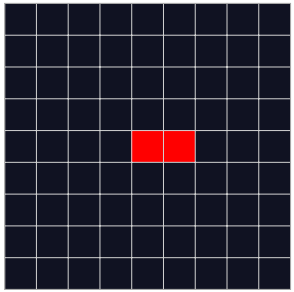
(zoomed in, a lot)

Your First Pixels (For Real This Time)

```
navy = (16, 18, 34)
img = [[navy] * 64 for _ in range(64)]

img[32][32] = (255, 0, 0)
img[32][33] = (255, 0, 0)

save_ppm(img, "first.ppm")
```

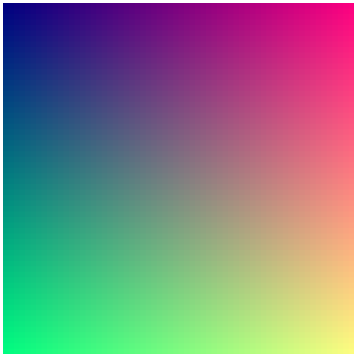


(zoomed in, a lot)

Congratulations: you are now a **graphics programmer**.

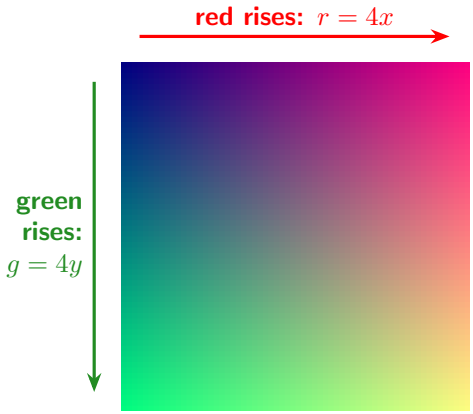
Now *All* the Pixels: Two Loops

```
1  img = [[(0, 0, 0)] * 64 for _ in range(64)]
2
3  for y in range(64):              # each row
4      for x in range(64):          # each pixel
5          img[y][x] = (4 * x, 4 * y, 128)
6
7  save_ppm(img, "setup_ok.ppm")
```



the color **depends on the coordinates**

Reading an Image Like a Graphics Programmer



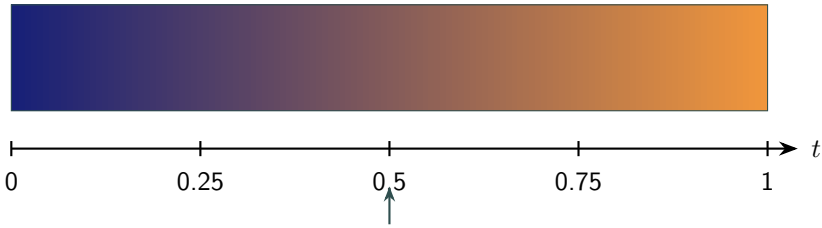
- red axis $\Rightarrow$ x grows rightward
- green axis $\Rightarrow$ y grows **downward**
- blue $= 128$ everywhere

Gradients

The Most Important Function in Graphics

How do you get from color A to color B... **smoothly**?

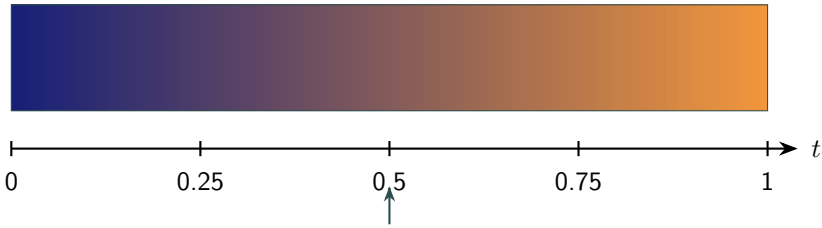
$A = (20, 30, 120)$ $B = (240, 150, 60)$



The Most Important Function in Graphics

How do you get from color A to color B... **smoothly**?

$A = (20, 30, 120)$ $B = (240, 150, 60)$



$$C = A + t(B - A)$$

"**lerp**" — linear interpolation

Lerp in Code

```
def lerp(a: float, b: float,
         t: float) -> float:
    """t=0 -> a, t=1 -> b."""
    return a + t * (b - a)

def lerp_color(ca, cb, t):
    return (int(lerp(ca[0], cb[0], t)),
            int(lerp(ca[1], cb[1], t)),
            int(lerp(ca[2], cb[2], t)))
```

Three lines — but it comes back **all semester**:

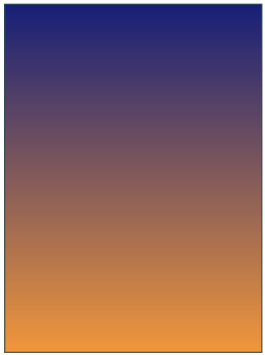
- gradients — today
- Bézier curves — week 2
- animation paths — week 2
- triangle filling — week 8
- texture filtering — week 12

Learn to *say* it: "lerp".

A Vertical Gradient: Lerp Once Per Row

```
H, W = len(img), len(img[0])
top = (20, 30, 120)       # night blue
bot = (240, 150, 60)      # dusk orange

for y in range(H):
    t = y / (H - 1)       # 0.0 .. 1.0
    c = lerp_color(top, bot, t)
    for x in range(W):    # whole row,
        img[y][x] = c
```



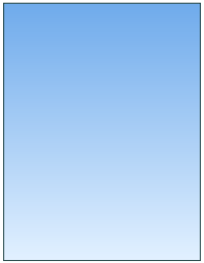
one color per **row**

Map the row to $t \in [0, 1]$, lerp the color, paint the row. That's a sky.

60 Seconds of Art Direction

The same ten lines, different endpoints:

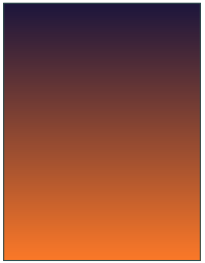
(110,170,235)



(225,240,255)

noon

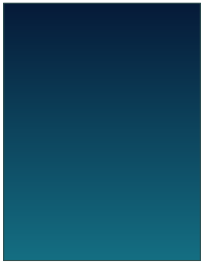
(25,20,60)



(250,120,40)

sunset

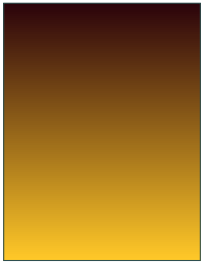
(5,25,55)



(20,110,130)

deep sea

(40,0,10)

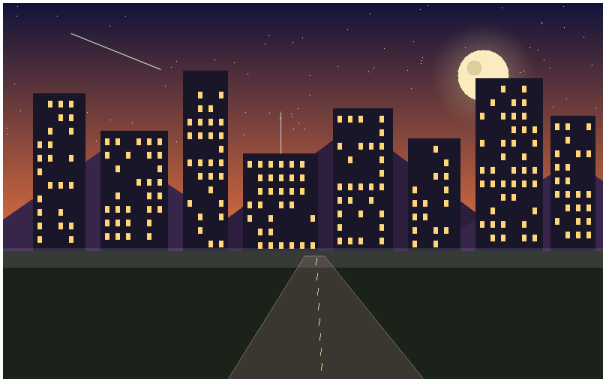


(255,200,40)

inferno

Assignment 1

A1: A Scene from Scratch — Out Friday



instructor exemplar — built with *only* the A1 toolkit

Part A — build the toolkit (70 pts)

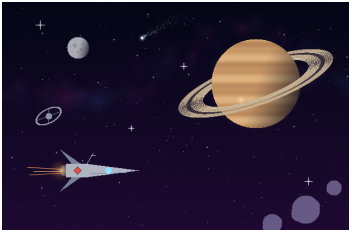
- pixels, rects, **gradients** today
- alpha blending Friday
- Bresenham lines Wednesday
- circles, flood fill Friday

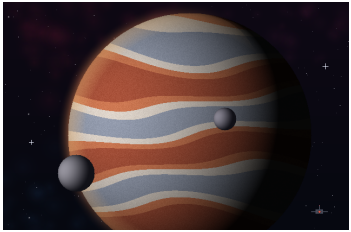
Part B — your own scene (30 pts)

- made only with *your* toolkit
- graded on craft, not just boxes

Due: Friday 11 Sep, 11:59 pm
Hard deadline • worth 3.6%

Yes, Really — Only Pixels and Arithmetic





Every one: `img[y][x]`, loops, and the ideas from this week.

What will *yours* look like?

Exit Ticket

Today's questions *(2 minutes)*

(a) On the image, which corner is pixel $(0, 0)$?

(b) What is the lerp of colors $A = (0, 0, 0)$ and $B = (200, 100, 50)$ at $t = 0.5$?

CS 453 — Exit Ticket #1

Name: _____

Roll no: _____

(a) _____

(b) _____

hand in on your way out

Wednesday: how to draw a *line* when all you have are square pixels.