Python Graphics Engine!

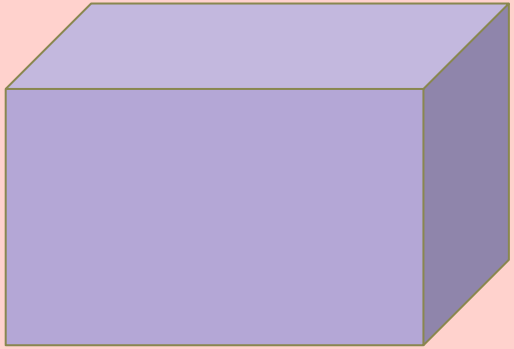
Thu, Mar 14, 2024
Raven (Ruiwen) Tang
Stuyvesant High School Spring 2023
Computer Graphics - Mr. Dyrland-Weaver

To get started... what is a graphics engine?

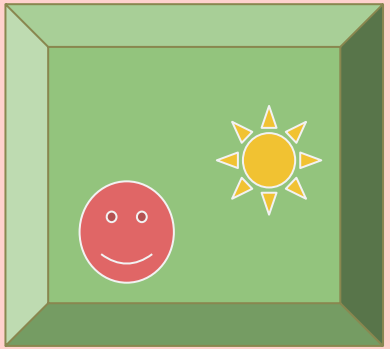
Take in commands from a user (who doesn't need to know the math or code behind it all!)

On a **canvas** of **size 100 x 100**, please give me a **smiley face** at **(20, 20)** and a **sun** at **(80, 50)**

** our graphics engine!! **



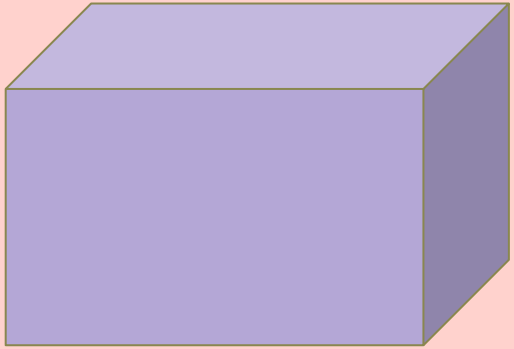
Produce pretty pictures and cool effects that hopefully align with what the user envisioned and exactly align with what the user told us!

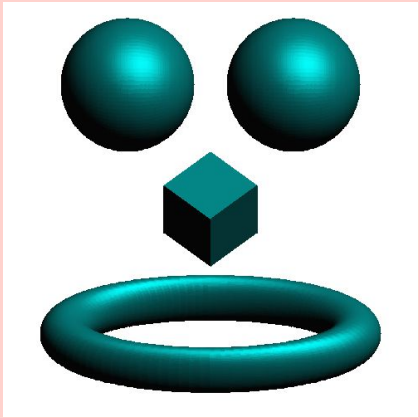


To get started... what is a graphics engine?

```
push
move
250 250 0
sphere
-100 150 0 80
sphere
100 150 0 80
push
rotate
x 45
rotate
y 45
box
-40 40 40 80 80 80
pop
push
move
0 -150 0
rotate
x 30
scale
1 1 0.5
torus
0 0 0 30 175
display
save
face.png
```

** our graphics engine!! **





Why?

- Other than being really, really cool…
- Renders 2D or 3D graphics with the correct lighting, proportions, relative positioning, all without the user needing to know what's going on behind the scenes
- Outputting a series of frames in succession = animation!
- Can imagine uses in design/modeling (game, infrastructure, architecture), simulation, virtual worlds

Approach to building engine from scratch

00 GETTING SOMETHING TO SHOW UP

01 ESTABLISHING TWO DIMENSIONS

02 EXPANDING TO THREE DIMENSIONS

03 COORDINATE SYSTEMS

04 FILLING FACES IN

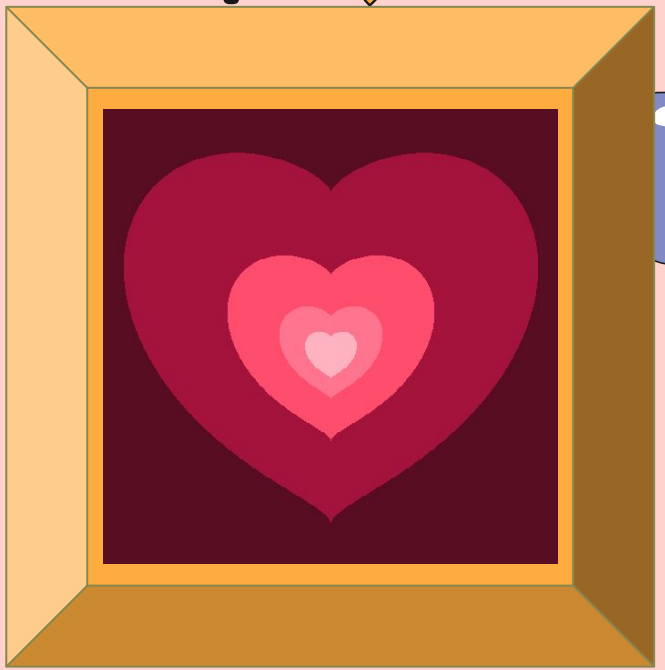
05 LIGHTING

06+ EXCITING EXTENSIONS

- Building a *mathematical foundation* in vector and matrix properties + operations
- Learning & implementing appropriate *algorithms* (the *connecting links* between the math and the visuals!)
- *Property-by-property development*, a working prototype at each stage (minimum viable product → more and more viable products)
- *Experimenting* with what can be produced at each stage (demo gallery!)
- *Reviewing* with instructor and peers, and *tweaking* before progressing to next stage

Step 0: Getting Something to Show Up

- *Problem:* How do we represent an image numerically or in a file?
- *Answer: Representing an image file* as a Portable PixMap: dimensions of the canvas, each pixel is represented as an *RGB triplet* in a *two-dimensional array*
- Introduces *color* and builds the blocks for the rest of our project!
- *Parsing* user's submitted script file



Step 1: Establishing Two Dimensions

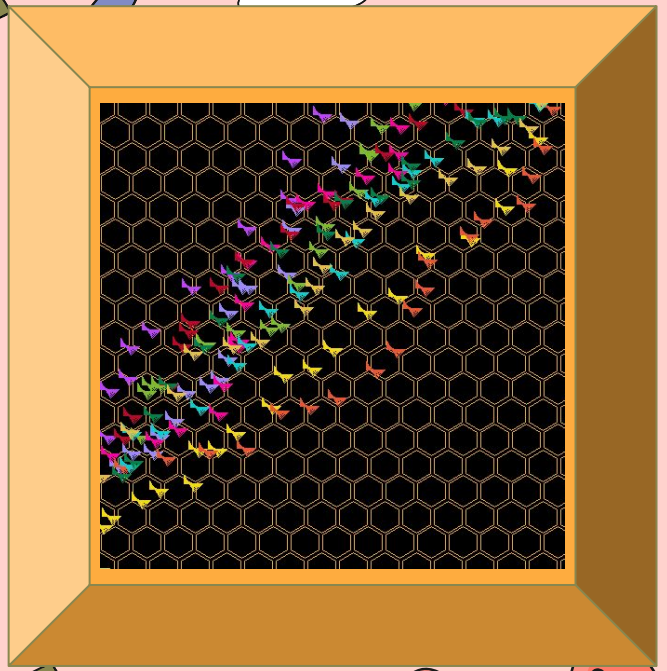
Step 1.1: Lines

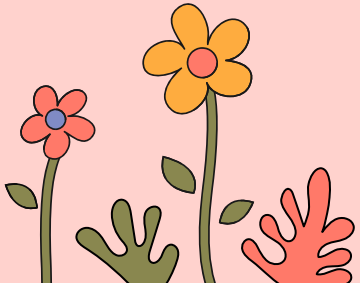
- *Problem:* a lines can have many different slopes, and most points on it are not going t exactly line up to integer coordinates → how do we decide which pixels to fill in?
- *Answer: Bresenham's Line Algorithm!*
- Introduces the idea of breaking the coordinate plane into octants for *casing, testing*

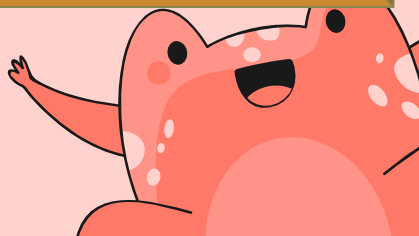
Step 1: Establishing Two Dimensions

Step 1.2: Edges

- *Problem:* How do we keep track of all of our lines?
- *Answer:* Store edges as points (each pair in the *edge list* is an edge, and each shape is stored as a series of edges)
- We have a *draw_line function* that can take in points from the edge list/matrix and draw them!



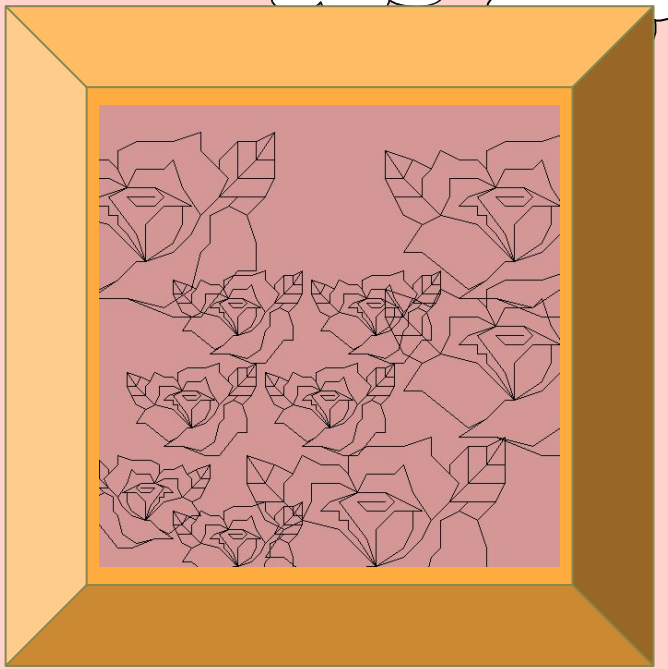




Step 1: Establishing Two Dimensions

Step 1.3: Transformations

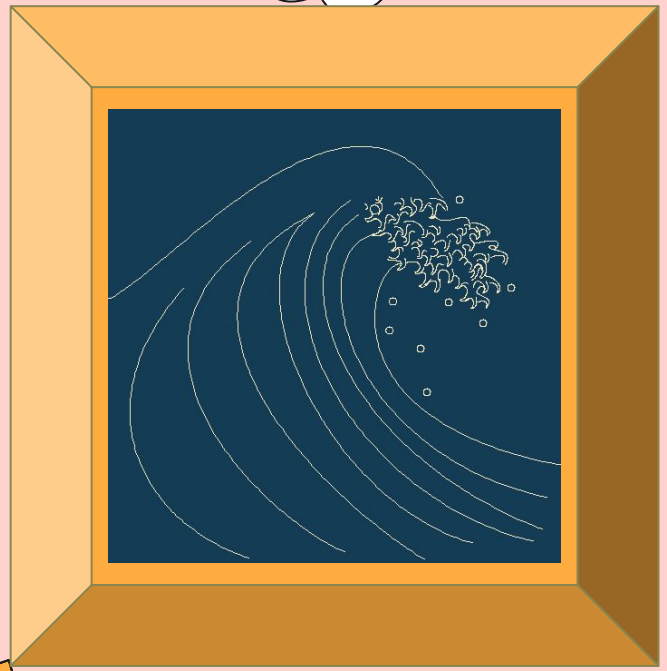
- *Problem:* What if I'm lazy but I want to make a slightly modified duplicate of part of my image?
- *Answer:* Allow user to *translate, dilate, and rotate* the edges they have!
- Keep track of a transformation matrix that can be applied to the edge matrix



Step 1: Establishing Two Dimensions

Step 1.4: Curves

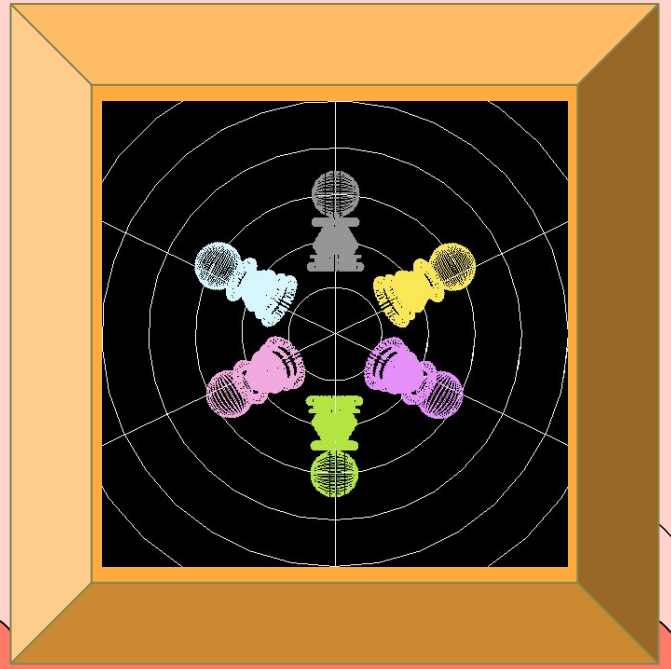
- *Problem:* How do we represent curves in terms of lines?
- *Answer:* define curve *parametrically*, and draw it with very many lines
- Types of curves available for the user to draw: *circles, splines, hermite curves, bezier curves* → can be integrated into our current matrix system



Step 2: Expanding to Three Dimensions!

Step 2.1: Rectangular Prism, Sphere, Torus

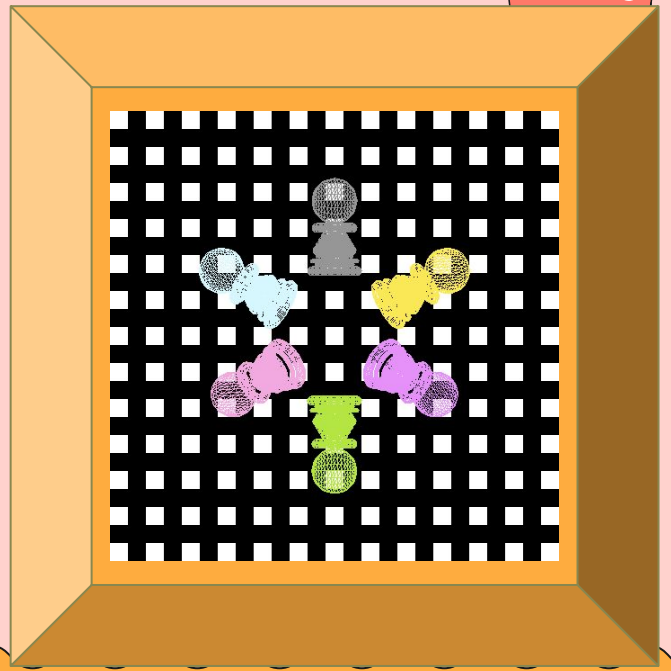
- *Problem:* How do we represent 3D shapes in terms of 2D shapes?
- *Answer:* utilize our lines and circles, and rotate them, keeping track of *z-coordinates* for points!
- Types of 3D shapes available for the user to draw: *rectangular prisms, spheres, toruses*



Step 2: Expanding to Three Dimensions!

Step 2.2: Compose Everything From Triangles

- *Problem:* Lines cannot capture *surfaces*
- *Answer:* Change our *basic unit of drawing* from a line to a *triangle*! Use a polygon list (every 3 points is a triangle) instead of an edge list
- Allows us to conceptualize which direction polygons are *facing* and *fill in* surfaces

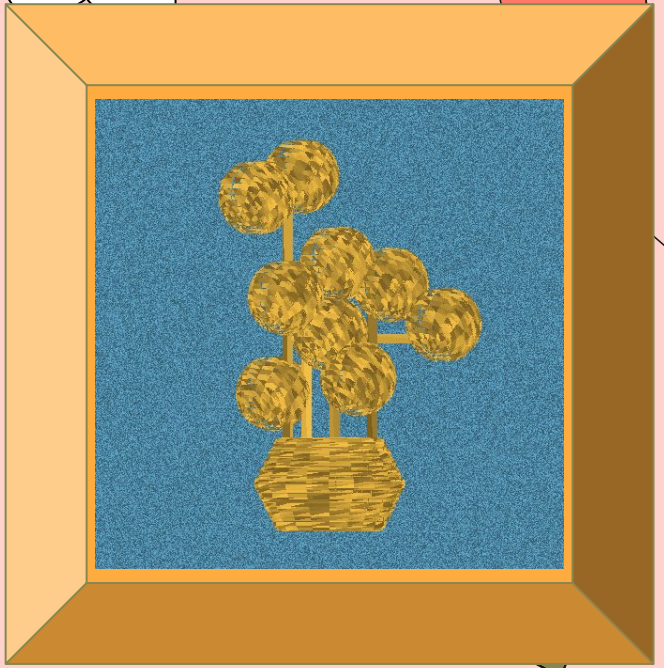


Step 3: Coordinate Systems

- *Problem:* What if I don't want to do everything relative to the *origin*? What if I want to do something *relative* to a shape I've already drawn?
- *Answer:* Shift from a *global coordinate system* to a *relative coordinate system*, where we *transform the world* rather than *transforming the shapes*
- Allows us to set up *shape dependence and independence*

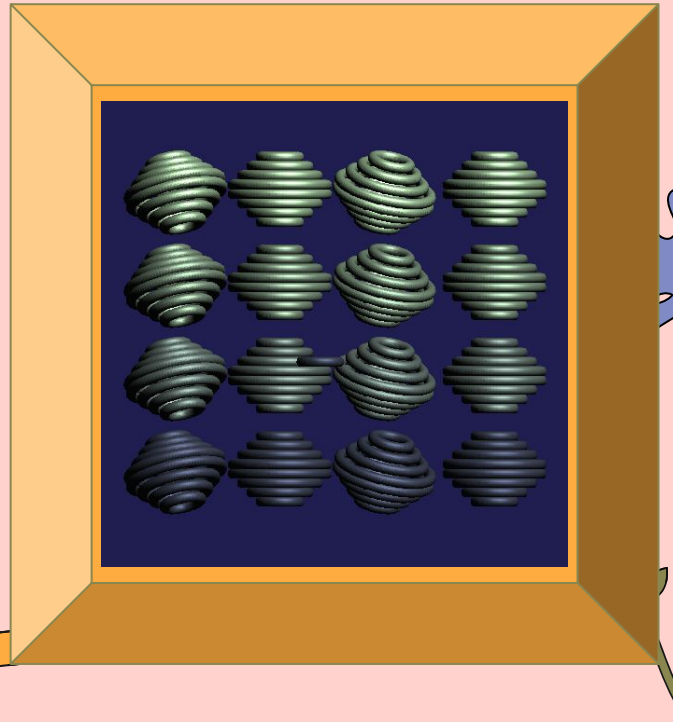
Step 4: Filling Faces In

- *Problem:* We want to add *color* to our surfaces (instead of a *mesh* sphere, I want a *solid colored* sphere)!
- *Answer:* Fill in each triangle by drawing a series of horizontal or vertical lines across the surface (*scanline conversion*)
- Fun additional challenge to handle: Check for surfaces that *face away* from the user and surfaces *covered* by other surfaces to reduce exhaustion



Step 5: Lighting

- *Problem:* 3D shapes shouldn't be one flat color!
- *Answer:* Implement *lighting* (*ambient* and *point*) and *reflection* (*diffuse* - matte, *specular* - glossy) by considering what colors a surface reflects, what colors the light sources emit, and *where the viewer is* (lots and lots of vector math)
- Fill in each triangle by calculating what color it should be

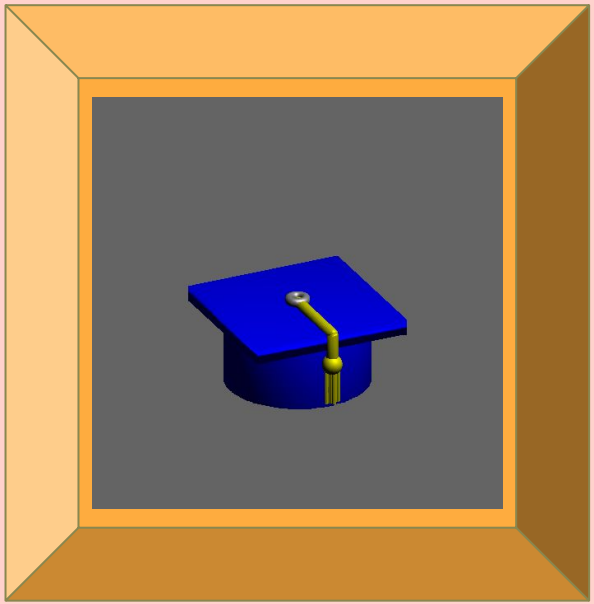


Step 6+: Exciting Extensions

Problem: I want to do more!

Answer(s) (possible):

- more lighting and shading models
- more shapes the user can ask for
- saving coordinate systems
- animations



The Big Picture

WHAT WE ACHIEVED

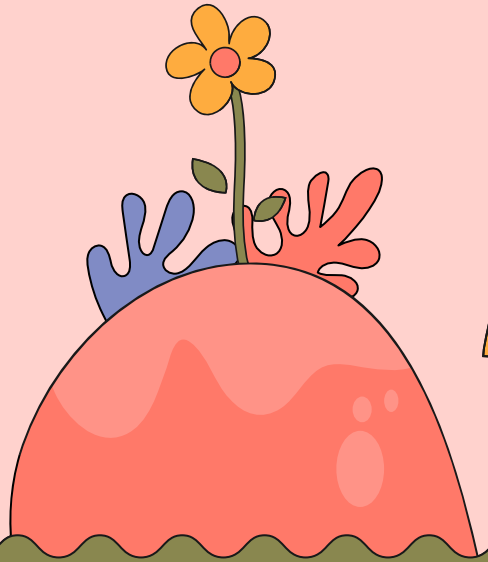
A graphics engine that can *take in a script* from a *non-technical user* and produce what they've asked for, handling *lines, 2D shapes, surfaces, lighting, transformations, and relative coordinate systems*

HOW AND WHY, AGAIN?

= understanding of math +
implementation of algorithms +
visions for potential outputs +
lots of testing +
loads of debugging +
constructive feedback

☆

Useful to any task or role with visual outputs!



Lessons Learned

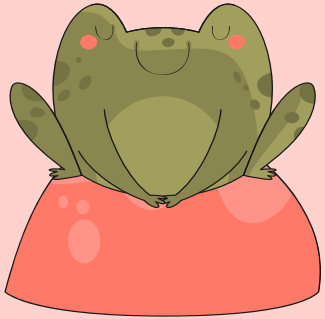
Technically:

- An *application* of *vectors and matrix algebra* to something that can literally be seen in the real world!
    - The math behind it all
    - How to *translate between* visual properties, math, and code

But more importantly, finding value in:

+ *Breaking it down* (what are the building blocks? leading with guiding problems)
+ *Mentorship* (guidance, feedback, someone who's done it before!)
+ *Pair programming* (bouncing off of each other, questioning + explaining, improving coding skills but also developing skills for articulation and critical evaluation of code)
+ *Gallery walks* (exposure to diversity of thought, implementation, production)
+ *Flexibility* (making choices based on memory & time, incremental development, revising features to suit our adapting needs, same project completed in any language!)
+ *Application* (seeing the system you worked on actually input and output cool stuff!)

Thank You :) *Questions?*



CREDITS

This project was developed under the excellent instruction, curriculum, and feedback of *JonAlf Dyrland-Weaver*

★

Step 6 (extensions) of this project was developed via pair programming with *Daphne Qin*, current student at Rice University

★

This presentation template was created by *Slidesgo*, including icons by *Flaticon* and infographics & images by *Freepik*