Lecture 2

Read Shiffman Chpts. 1 and 2

Writing Clean Code

Syntax

- Computers are a bit like an uptight grammar teacher
  - If everything is not stated precisely, they will not understand you
- Really, computers are stupid!
  - Computer design is quite brilliant
  - Difficult to make computers understand ambiguity

Syntax

- Semicolons end a command

  e.g. rect();

Syntax

Braces

- ( ) for commands

  e.g. rect();
- { } for blocks of code

```
void draw()
{
        //commands
}
```

- [ ] for arrays (coming later)

(Human) Readability

- Whitespace
  - Leave blank lines between blocks
- Comments
  - Helps you and others to read and understand code
  - //Single line
  - /* multiple lines

    line 2*/

(Human) Readability

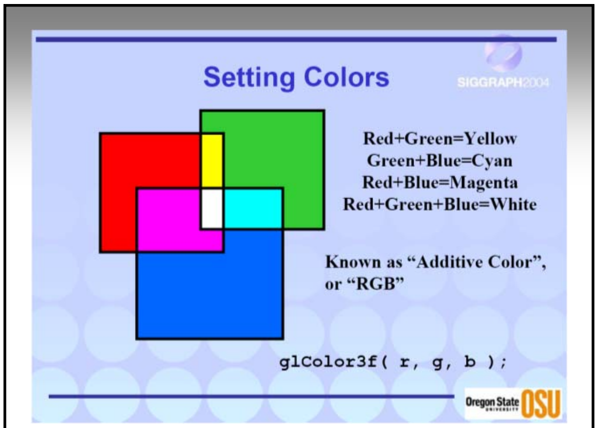
- Indent all blocks for readability (4 spaces)
  - edit->autoformat

```
void setup()
{
    stroke(4);
    for(int i = 0; i<10; i++)
    {
        //do something in a loop
    }
}
```

Color

What are the primary colors?

- It depends…
- Subtractive primaries:
  - Cyan, Magenta, Yellow
  - e.g. used for print
- Additive primaries:
  - Red, Green, Blue
  - Used when mixing light e.g. a computer display



Setting Colors

SIGGRAPH2004

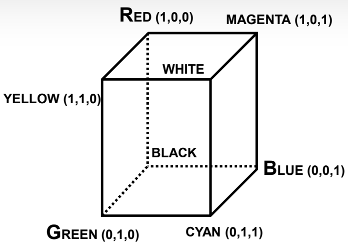
Red+Green=Yellow
Green+Blue=Cyan
Red+Blue=Magenta
Red+Green+Blue=White

Known as "Additive Color", or "RGB"

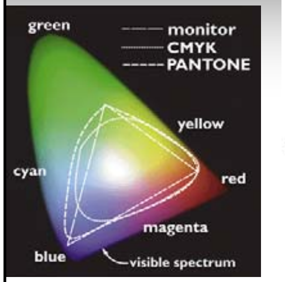
```
glColor3f( r, g, b );
```

Oregon State OSU

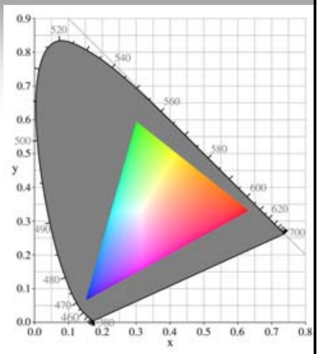
RGB Colour Model



Color Gamut



Source: American Institute of Physics



Source: Wiki Commons

Convention in Notes

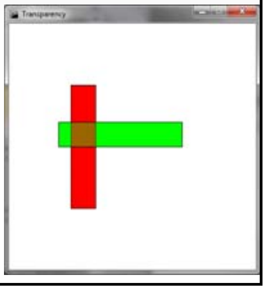
- <name> indicates a value you must specify for a command
- e.g. line(<startX>, <startY>, <endX>, <endY>);

Coding Color in Processing

- Color is defined by a tuple (<R>, <G>, <B>)
- 0 is none of a color
- 255 is max color
- Examples:
  - Bright Red: (255, 0, 0)
  - Bright Yellow: (255, 255, 0)
  - Dull Yellow: (100, 100, 0)
  - Mid Grey: (120, 120, 120)
  - e.g. fill(0,0,200); //To draw mid blueshapes

Alpha

- Fourth parameter that defines transparency
- (<R>, <G>, <B>, <A>)
- 0 transparent
- 255 is opaque
- 255 is default value



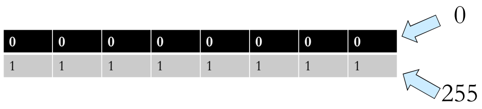
Why 255?

- Computers represent all data combinations of bits
- *Bit* can be 0 (off/false) or 1 (on/true)
- Numbers represented by multiple bits

2^2 = 4	2^1 = 2	2^0 = 1	#
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Bits, Bytes and Pretzels

- Computer hardware designed to work with particular “group sizes” of bits:
  - 4, 8, 16, 32, 64
- 1 *Byte* is 8 bits
- 1 Byte can hold 2^8 = 256 values
  - 0 – 255



0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1

Moving Objects

- This is another form of state
  - *transformation state*
- translate(<x>, <y>);
- rotate(<angle>);
  - <angle> must be in radians (more on this later)
  - For now, just use angles in degrees and wrap with the radians() method
  - e.g., to rotate 20 degrees, use rotate(radians(20));
  - Pivot is the relative origin of the object
    - i.e. the point the <x>, <y> offset in say rect() is applied from

Saving Images

- save(“image.jpg”);
  - Can associate with a mouse click or button press

```
void mousePressed()
{
        save(“myImage.jpg”);
}
```

Saving Images

- To save a sequence:
- saveFrame(“image###.jpg”);
  - ### will be automatically replaced by the image number
  - Can add to draw

More on Transformation State

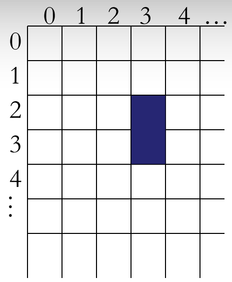
Transformation State

- If you think of “stroke” as setting the color of pen that an outline is drawn with, similarly translate and rotate set the state of the origin
  - translate updates the position of the origin
  - rotate updates the orientation of the origin
- By default, the origin is the upper left corner
  - x increases left to right
  - y increases as you move down
- translate and rotate update this
  - All later commands are effected

Problems

- translate(,);
- fill(0, 0, 255);
- rect(0, 0, 100, 200);

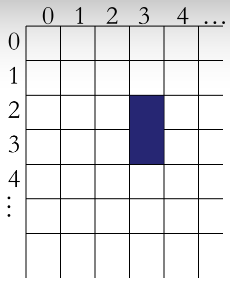
- translate(100,100);
- translate(,);
- fill(0, 0, 255);
- rect(0, 0, 100, 200);

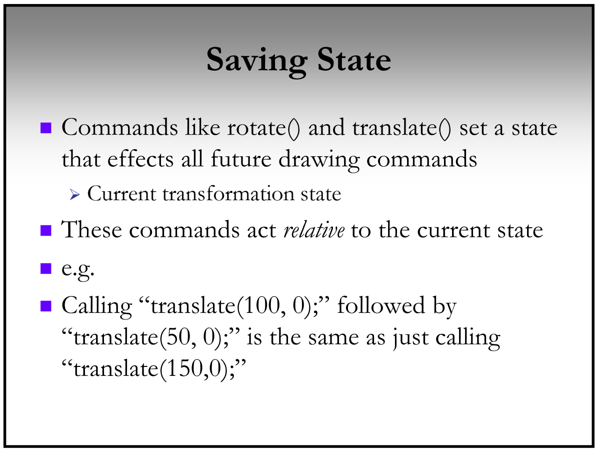


Problems

- translate(,);
- rect(200, 100, 100, 200);

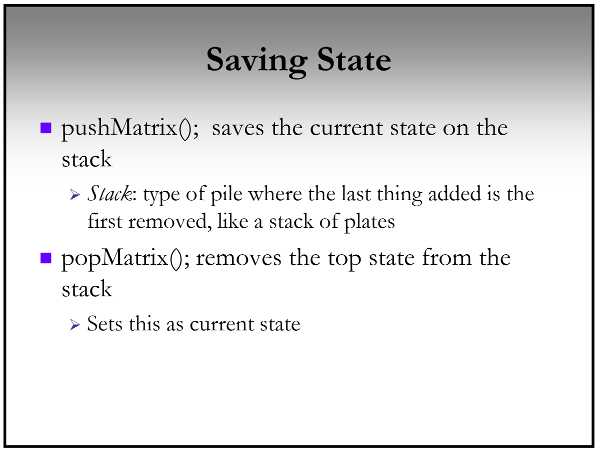
- translate(,);
- rotate(radians( ));
- rect(0, 0, 200, 100);





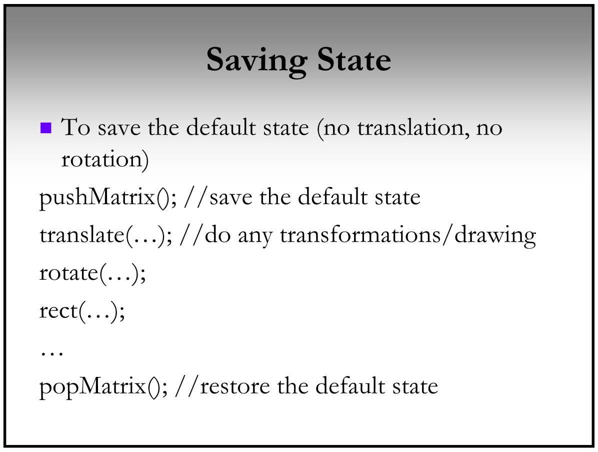
Saving State

- Commands like rotate() and translate() set a state that effects all future drawing commands
  - Current transformation state
- These commands act *relative* to the current state
- e.g.
- Calling “translate(100, 0);” followed by “translate(50, 0);” is the same as just calling “translate(150,0);”



Saving State

- pushMatrix(); saves the current state on the stack
  - *Stack*: type of pile where the last thing added is the first removed, like a stack of plates
- popMatrix(); removes the top state from the stack
  - Sets this as current state



Saving State

- To save the default state (no translation, no rotation)

```
pushMatrix(); //save the default state
translate(…); //do any transformations/drawing
rotate(…);
rect(…);
…
popMatrix(); //restore the default state
```