Intro to Project Iteration #1: "Brushwork"

CSCI-3081: Program Design and Development

Class Reset: Where are we in the semester? How does this all fit together?

- Class Meetings and Topics
- Weekly Writing Assignments
- Lab Sessions
- Project
- Exams

Brushwork demo from the TAs.

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

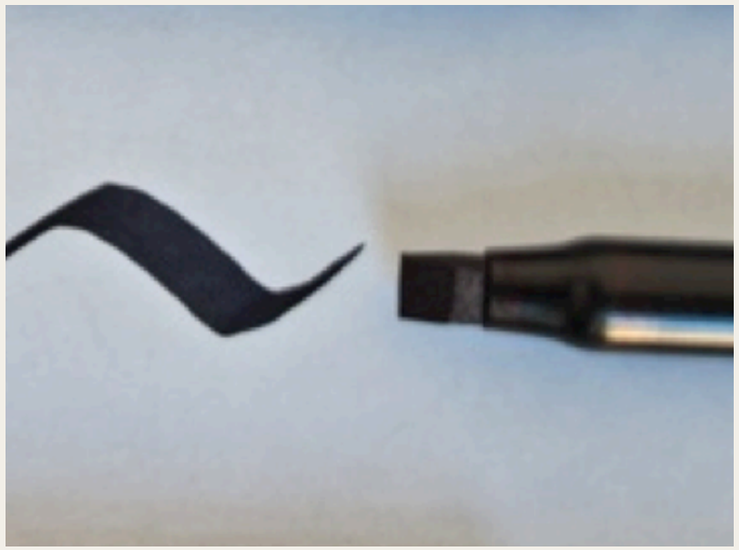
General Requirements

1. When Brushwork is running, there should always be exactly one active tool.
2. The active tool should apply itself to the digital canvas when the mouse is clicked and dragged in the CanvasWindow.
3. All of the tools (except the Eraser) should take the current tool color into account when they are applied to the PixelBuffer.
4. The program needs to run fast enough to make the tools feel responsive to the user.
5. To help the program run as fast as possible, each tool must precompute and store its 2D shape in a small array in memory called a “mask”. (We’ll come back to this one…)

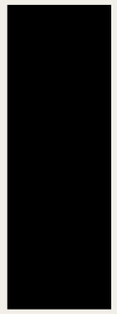
Pen Tool

- Its mask should be small circle.
- It should be completely opaque.
- Color set interactively using the Tool Window GUI.

Calligraphy Pen Tool



- Mask should be a rectangle that is taller than it is wide:

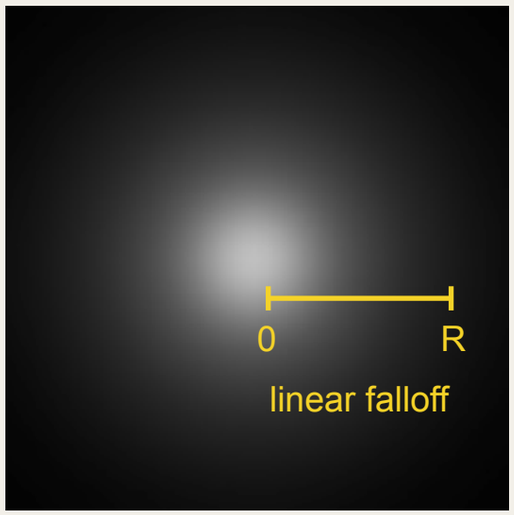


- Completely opaque.
- Color set interactively using the Tool Window GUI.

Highlighter Tool

- Same rectangular shape mask as the Calligraphy Pen.
- But it is semi-transparent.
  - The color applied to the canvas should be 40% the color of the highlighter and 60% whatever color is already on the canvas (or the background color if no color has yet been applied).
- Color set interactively using the Tool Window GUI.

Spray Can Tool

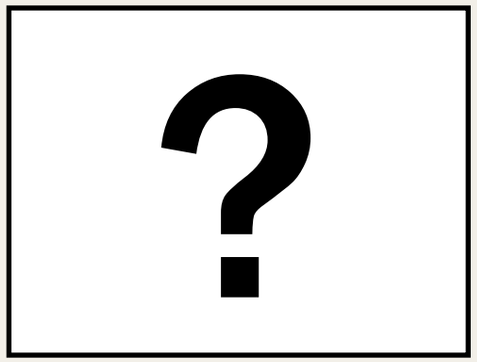


- Special mask that is circular in shape but has a linear falloff to mimic the dispersion of color from the spray.
  - At the centermost pixel, the color is strongest. Here, the color applied to the canvas should be 20% the color of the spray can and 80% whatever color is already on the canvas.
  - At a distance of R (spray radius) pixels from the center, the spray should have no effect, that is, the color should be 0% the color of the spray can and 100% whatever color is already on the canvas.
  - At distances between 0 and R pixels from the center, the amount of color should change linearly between these two values.
- Specific color to spray set interactively using the Tool Window GUI.

Eraser Tool

- Circular mask.
- Resets the pixels underneath the mask to the original background color of the canvas.
- Cannot simply hardcode your eraser to always draw a solid white circle.

“Special” Tool

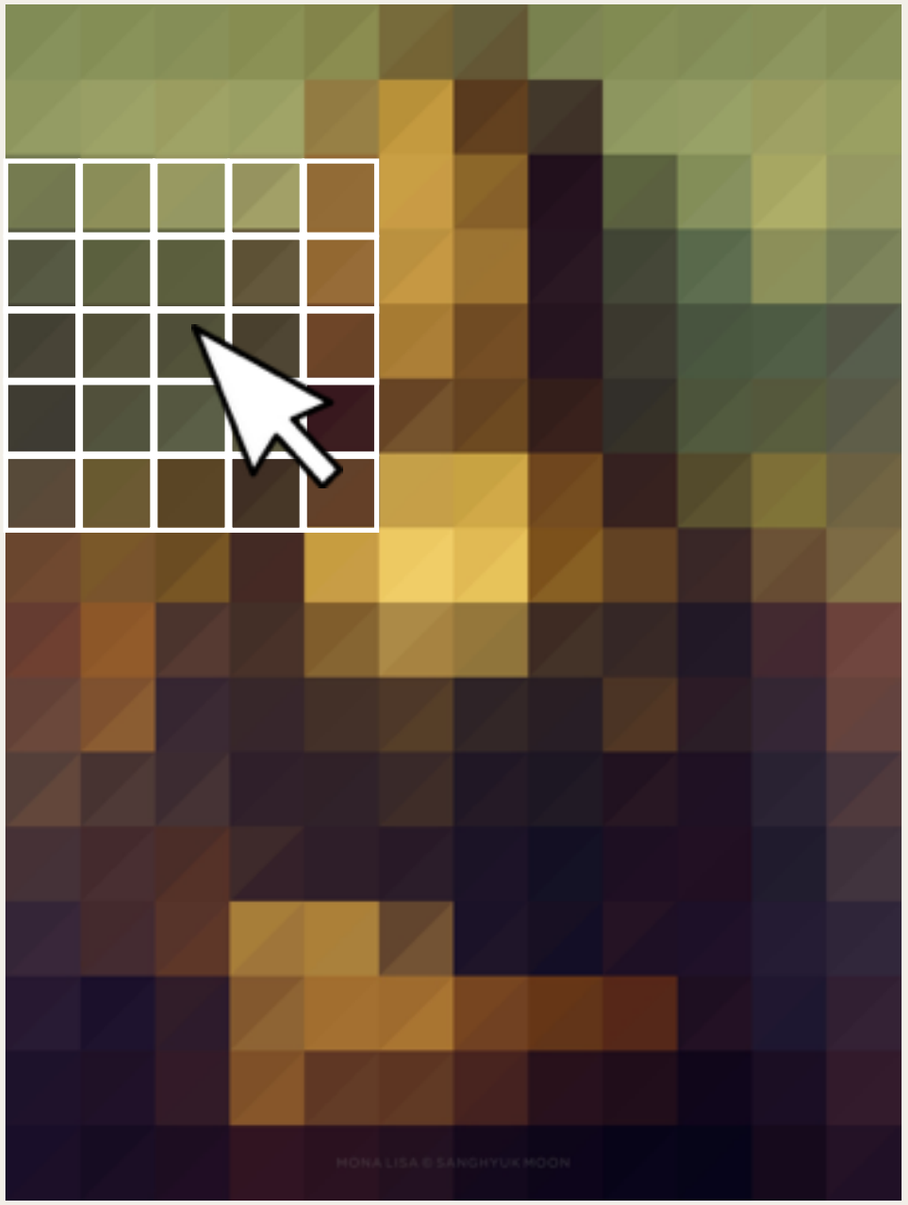


- The final tool is left for you to develop on your own!
- Watercolor brush?
- Crayon?
- …

What is a Mask?

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

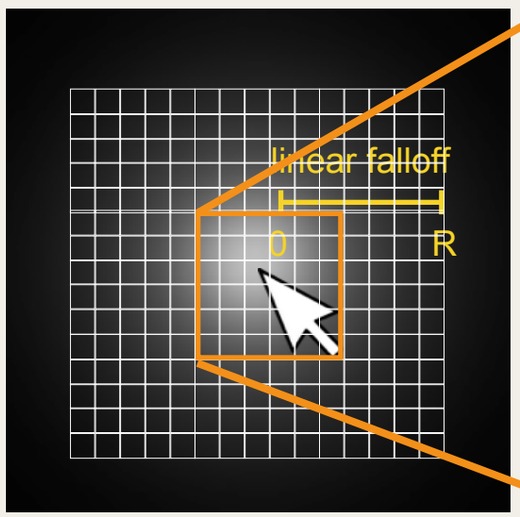
A mask is a 2D array of floats.

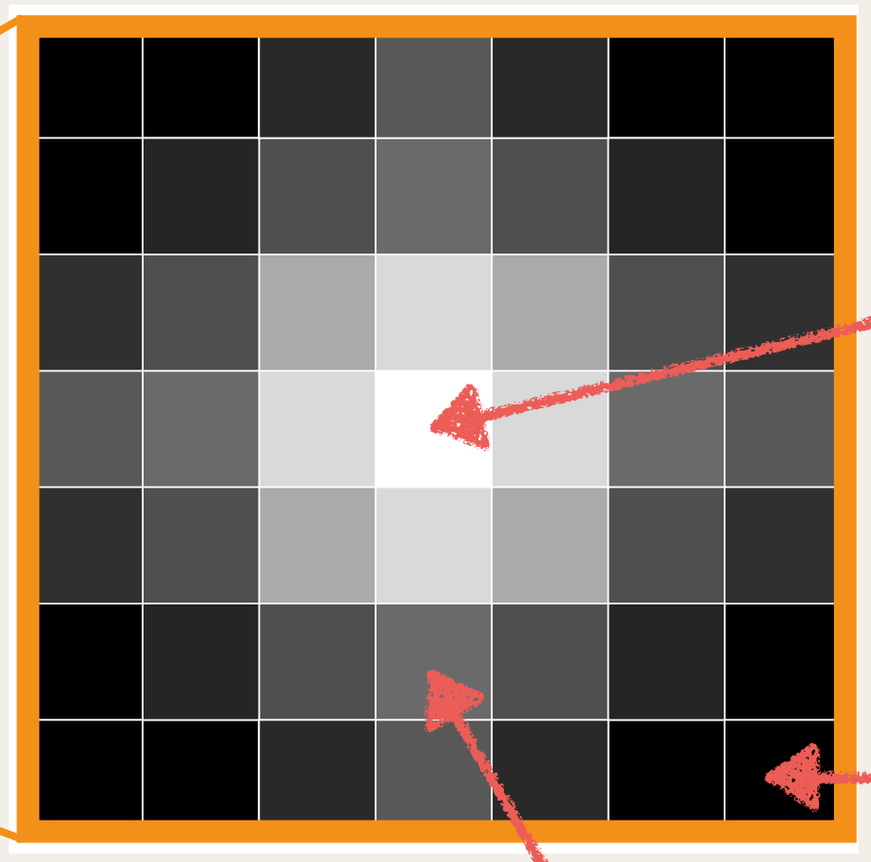


zoomed in view of our canvas

- Like an invisible “window” attached to the mouse cursor that slides over the canvas as the mouse moves.
- Stores a pre-computed “footprint” for each tool so that we don’t need to recompute this footprint every time the mouse moves.

The mask stores the “amount of influence” the tool should have on the canvas for each pixel.





1.0 = highest influence; new color on the canvas is entirely determined by the brush color.

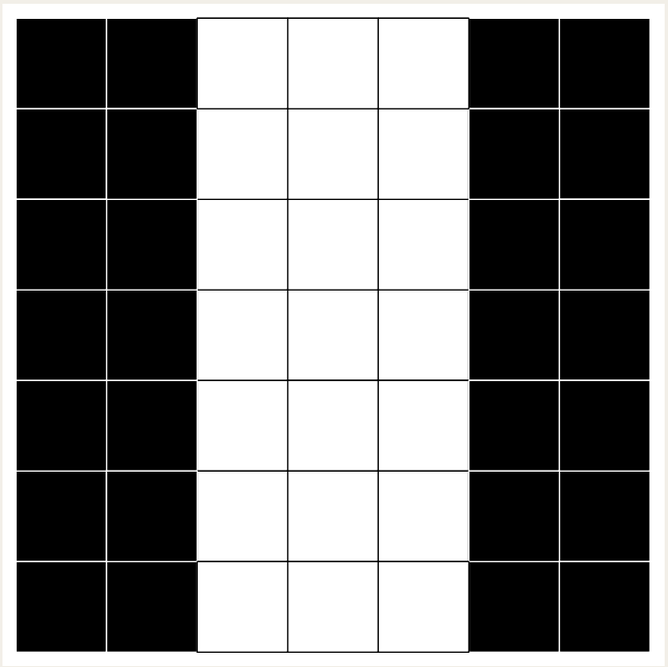
0.0 = no influence; the canvas color is unchanged

0.4 = some influence; the canvas color is 40% from the brush + 60% from the canvas.

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

Other masks?

- You can create many cool tools just by changing the mask.
- What would a mask be for the calligraphy pen?



(if you assume all masks are square)

Brushwork: Your Upcoming Assignments…

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

This Week

- Read the official software requirements document on Moodle.
- Your Weekly Writing assignment is to extend last week's UML Class Diagram to now include a diagram of ***your idea*** for how to design the Brushwork program.
- (Note, this is not yet group work… I want you all to go through this design exercise individually!)
- During Friday's lab, you'll meet your project team.

Next Week

- Plan for one of your first team meetings to be a discussion of your UML diagrams.
- Your group will need to settle on one design, or create a new design together that uses the best ideas from everyone in your group.
- Start working as a team to create Brushwork!

Resources on Moodle

Under the Lab/Project Heading:

- **Project Iteration #1 (Brushwork) Official Software Requirements**

Under the Weekly Writing Heading:

- **Weekly Writing UML Diagramming Assignment**

Now, Switching Gears:

C++ Technical Detail on Pointers (Round 1)

UNIVERSITY OF MINNESOTA
PROFESSOR DANIEL F. KEEFE

Last time, I asked you to review some example code that uses pointers to C++ classes… why important? questions?

```
// EXAMPLE 6:
// This strategy of working with pointers to a base class is especially
// useful when storing a long list of Ducks.  We'll use the C++ std::vector
// class to create a list of pointers to Ducks.

std::vector<Duck*> duckList;

Duck *duck1 = NULL;
duck1 = new MallardDuck();
duck1->setName("Mallard Junior");
duckList.push_back(duck1);
```

```
// If we want, we can skip the first step of setting the pointer to NULL
// and write the code more compactly like this:
Duck *duck2 = new RubberDuck();
duck2->setName("Squeaky");
duckList.push_back(duck2);

Duck *duck3 = new DecoyDuck();
duck3->setName("Mr. Quiet");
duckList.push_back(duck3);

// Now, we can work with the whole list of ducks very easily
for (int i=0; i<duckList.size(); i++) {
    duckList[i]->fly();
    duckList[i]->performQuack();
}
```

In C++ you need to have a solid understanding of what's happening when you use pointers.

- This is so important that we're going to cover it in small chunks over several class meetings.
- Today we'll start with the basics:
  - C and C++ provide pointers. These allow for flexible data structures.
  - A pointer can be thought of as an address of a location in memory.

Pointers in C++

Consider:

```
int size = 100;
int* ps; // pointer to int
//int *ps, notapointer;
```

Defining ps

```
ps = &size; // ps points to size
```

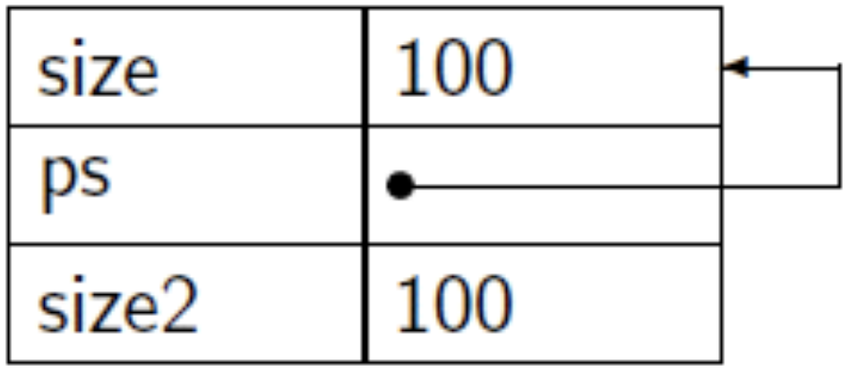
`&size` is the address where size exists in memory

Referencing the data ps points to

```
int size2 = *ps; // size2 == 100
```

Visualizing pointers

- A visual model of memory:



- Memory cells are labelled by names and store values.
- Some values are pointers to other cells.

Exercise

- Draw the model of memory to illustrate the state of memory after the following code fragment has executed.

```
char first = 'A' ;
char second = 'B' ;
char third = 'C' ;
char *p1 = 0 ;
char *p2 = 0 ;
char *p3 = 0 ;
p1 = &first ;
p2 = p1 ;
p3 = &second ;
*p1 = third ;
```

Pointers, evaluation

- The expression `ps` evaluates to the address held in `ps`.
- The expression `*ps` evaluates to the value stored in the address held by `ps`.
- `ps = 0;`
  - makes ps point to “nothing”.
  - No valid data can be stored at address 0.
  - This is “setting it to null”.
  - Can also write: `ps = NULL;`
- It is good programming practice to always initialize pointers to NULL:
  - `int *ps = NULL;`

Pointers, comparison

- If `ps == 0` then evaluating `*ps` has unspecified behavior, in other words “big crash”, “boom”.
- Thus, we also test that pointers are not equal to 0 before accessing the data they may point to.
- `if (ps != NULL) ... *ps ...`
- `if (ps) ... *ps ...`
- These are equivalent - C boolean operators work on integers, 0 = false, anything else = true.

```
int size = 100;
int *ps = 0;
int size2 = 0;
int *ps2 = 0;
ps = &size;
ps2 = &size2;
size2 = *ps;
if (size == size2)
  cout << "size == size2" << endl;
else
  cout << "size != size2" << endl;
if (ps == ps2)
  cout << "ps == ps2" << endl;
else
  cout << "ps != ps2" << endl;
if (*ps == *ps2)
  cout << "*ps == *ps2" << endl;
else
  cout << "*ps != *ps2" << endl;
```

What does this print out?

Arrays in C and C++

Arrays are implemented using pointers.

```
int x[5] = {1,4,9,16,25};
```

This allocates and initializes an array of size 5 and has `x` point to the first element of the array.

Indexing begins at 0, so

```
x[0] == 1, ..., x[4] == 25
```

Arrays and Short-Circuit Evaluation

Example: The following loop finds a value in x greater than 15 if it exists.

```
int i = 0;
int x [5] = {1,4,9,16,25};

while (i < 5 && x[i] <= 15) {
  i++;
}

if (i < 5)
  cout << "found one " << x[i];
else
  cout << "didn't find one";
```

Pointer Arithmetic

- This loop is equivalent but uses **pointer arithmetic**.

```
int i = 0;
int x[5] = {1,4,9,16,25};
int *y = x;
while (i < 5 && *y <= 15) {
   i++;
   y++;  // here we add 1 to the pointer
}
if (i < 5)
   cout << "found one " << *y;
else
   cout << "didn't find one";
```

Pointer Arithmetic

Adding 1 to a pointer, moves it to the next data object after it.

We assume that objects are stored contiguously.

We can add any integer value, as in `x + 4` above

`x[0]` is the same as `*x`.