

# **CS 248 Assignment 1**

## **Paint Program**

**Introduction to Computer Graphics Help Session**  
**presented by Georg Petschnigg**  
**Stanford University**  
**October 4, 2002**

# Session Overview



- **Getting Started**
- **Assignment Discussion**
  - Overpainting Brush
  - Tinting Brush
  - Brush Visualization
- **Grading Details**
- **Extra Credit**
- **Questions**

# Getting Started



- 1. Read assignment carefully**
- 2. Go to review session**
- 3. Familiarize yourself with Sweet Hall Lab  
Located in the Basement of Sweet Hall**

# Development Environment



- **Ways to work with TA Support**
  - Go to Sweet Hall Lab (Best Way – 5 Minutes)
  - Work Remotely (Good Way – 5 Minutes)
- **Ways to work without TA support**
  - Reproduce Sweet Hall Lab development environment on you own Machine (1 Hour)
  - Your code still has to work on the Sweet Hall machines (more risk for you)

# Sweet Hall



1. Pick a free computer, Log on
2. Copy assignment from  
`usr/class/cs248/assignments/assignment1/`  
to local directory
3. Run 'make'
4. Run './paint.i386-linux'

# Working Remotely



1. ssh to firebird, raptor or leland
2. Export the display
3. Follow instructions on previous slide

```
>ssh raptor1.stanford.edu  
>setenv DISPLAY yourIP:0.0  
>xterm &
```

**Demo: Using Sampson, PC-Leland, X-Win32, AFS Service on WindowsXP, make an run assignment..**

# Assignment Discussion



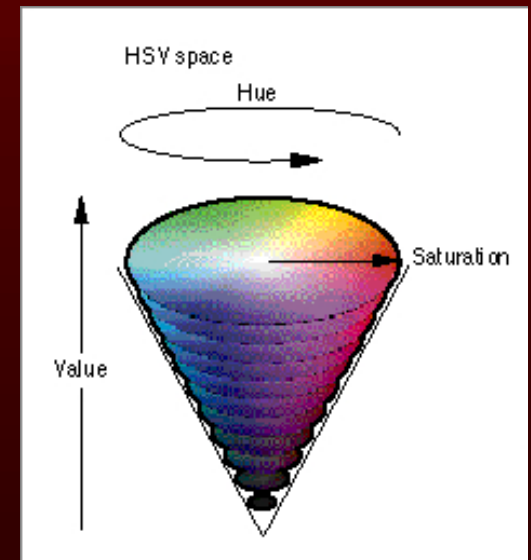
- **You are going to write a paint program**
  - Teaches you 2D Raster Graphics
  - Visualize concepts learned in Class (Brushes, HSV)
  - This assignment is a lot of fun
  - Be creative with extra credit
- **The next slides follow the Assignment (Handout #3) step by step**
  - Reminder: Read the assignment

# Part 1: Over Painting Brush



- **Rectangular Overpainting Brush**
  - Like Microsoft Paint or “Pencil Tool” in PhotoShop
- **Color Picker for RGB, HSV**
  - See [http://www.ewertb.com/java/Java\\_ColorPicker.html](http://www.ewertb.com/java/Java_ColorPicker.html) or any commercial Paint Program
    - Value (1.0 bright, 0.0 black)
    - Saturation (1.0 strong hue, 0.0 faded hue)
- **Size Control for Brush**

Demo: Painting, Picking Colors in PhotoShop



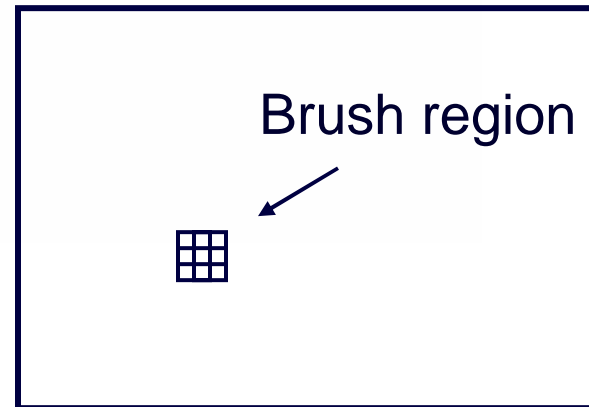


# Part 1: Basic Painting Loop



Basic painting loop:

```
do forever
  readmouse (x, y, button down)
  if button down then
    do i = - brushwidth/2 to brushwidth/2
      do j = - brushwidth/2 to brushwidth/2
        modify pixel (x, y, i, j)
      enddo
    enddo
  endif
enddo
```



# Part 1: Over Painting Brush



- Once you are done with Part 1 you should be able to draw some basic images



- Notice the hard edges and jaggies around the stroke... this is what Part 2 will fix

Demo: Show Jaggies using Snoop

## Part 2: Tinting Brush

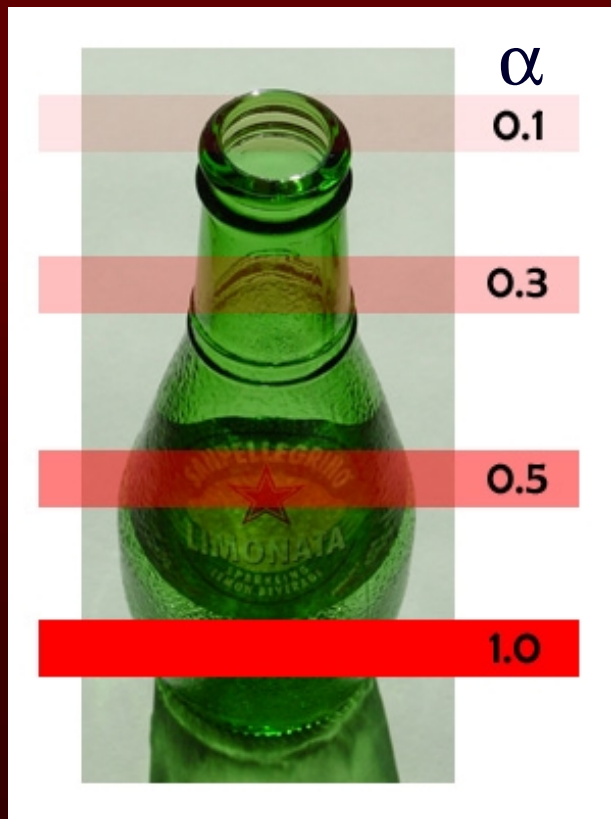


- **Implement Weighted Mask Driven Brush as described in Handout #4**
  - Instead of a rectangular brush, have it gently “blend” to its surroundings. Use HSV interpolation
- **Checkboxes for interpolating along H,S,V axis**
  - Allow all permutations HSV, HS, HV, SV, H, S, V
- **Choose a mask function and give use control over it**
  - Make sure it gradually falls off to zero at the edge!

# Part 2: Weighted Blending



Like painting with partially transparent paint.  
Commonly referred to as “alpha” blending.



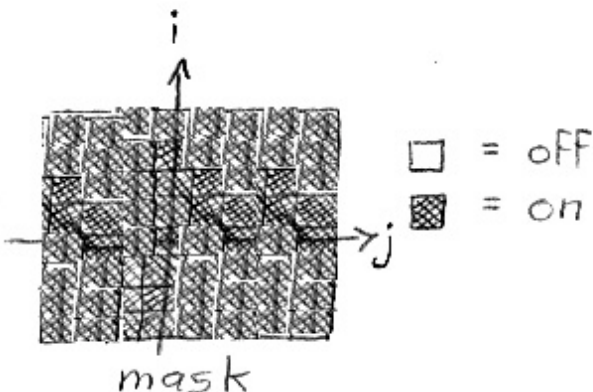
Compositing equation

$$C_{\text{new}} = (1-\alpha) C_{\text{old}} + \alpha C_{\text{paint}}$$

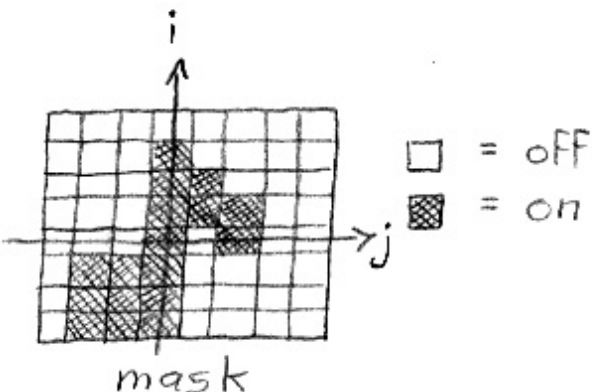
# Part 2: Mask driven painting



Lookup array determines how each pixel in the brush is affected.



Paint every pixel in the brush region



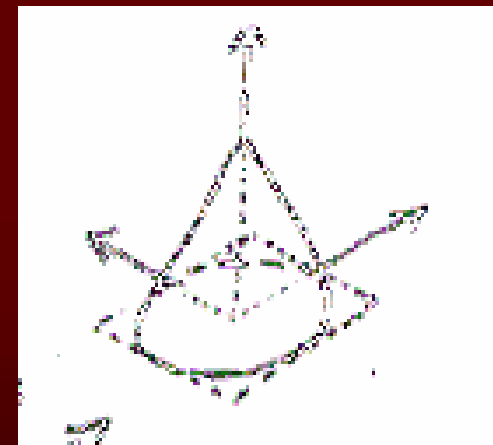
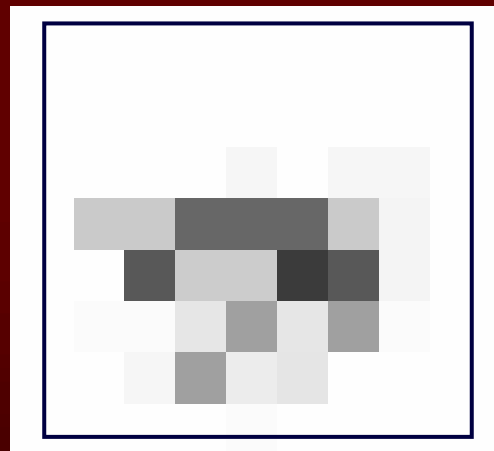
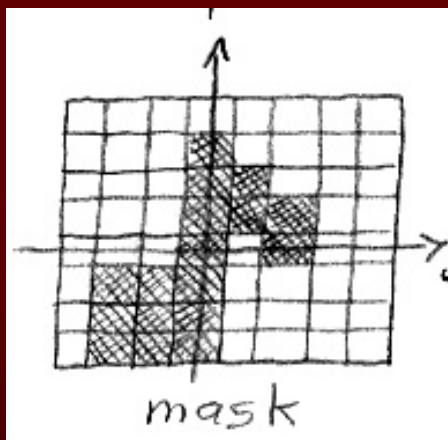
Paint only some of the pixels



## Part 2: Weighted mask driven painting



Mask contains alpha/weight for each pixel in brush



- $0 \leq \alpha \leq 1$  everywhere
- $\alpha$  is highest in the middle of the mask,  $\leq 1$
- $\alpha$  is smooth ( $\geq C^0$ ) except (optionally) at the center
- $\alpha$  falls off to zero at the edges of the mask

# Part 2: RGB vs. HSV interpolation



## RGB interpolation

$$\text{NewR} = (1-\alpha) \text{CanvasR} + \alpha \text{PaintR}$$

$$\text{NewG} = (1-\alpha) \text{CanvasG} + \alpha \text{PaintG}$$

$$\text{NewB} = (1-\alpha) \text{CanvasB} + \alpha \text{PaintB}$$

---

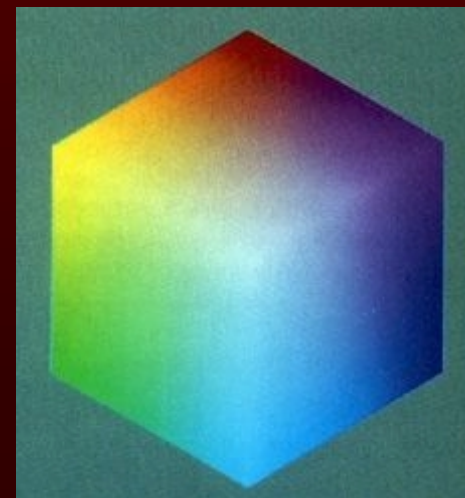
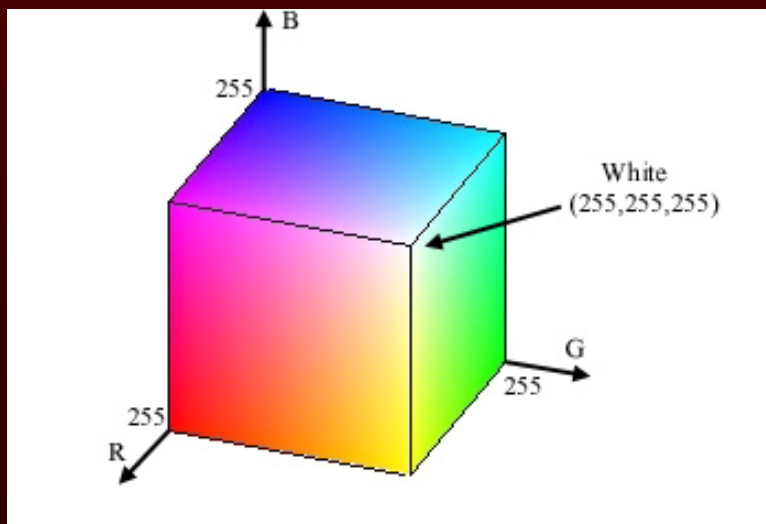
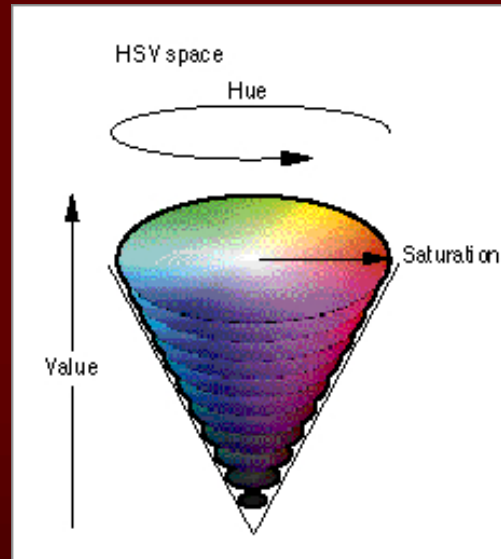
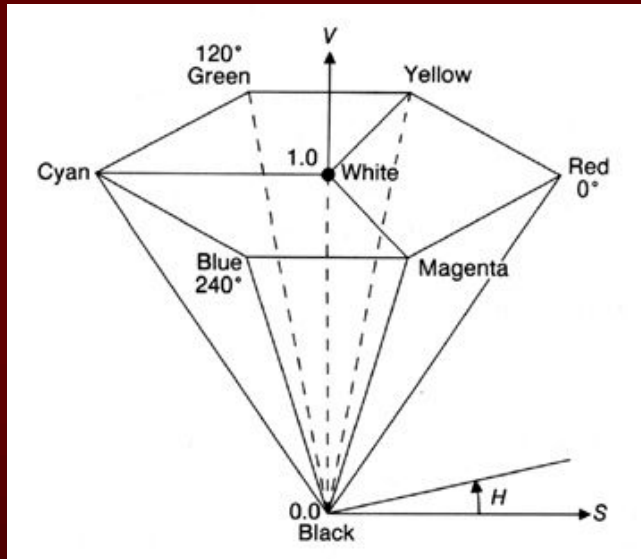
## HSV interpolation

$$\text{NewH} = (1-\alpha) \text{CanvasH} + \alpha \text{PaintH}$$

$$\text{NewS} = (1-\alpha) \text{CanvasS} + \alpha \text{PaintS}$$

$$\text{NewV} = (1-\alpha) \text{CanvasV} + \alpha \text{PaintV}$$

# Part 2: RGB vs. HSV interpolation



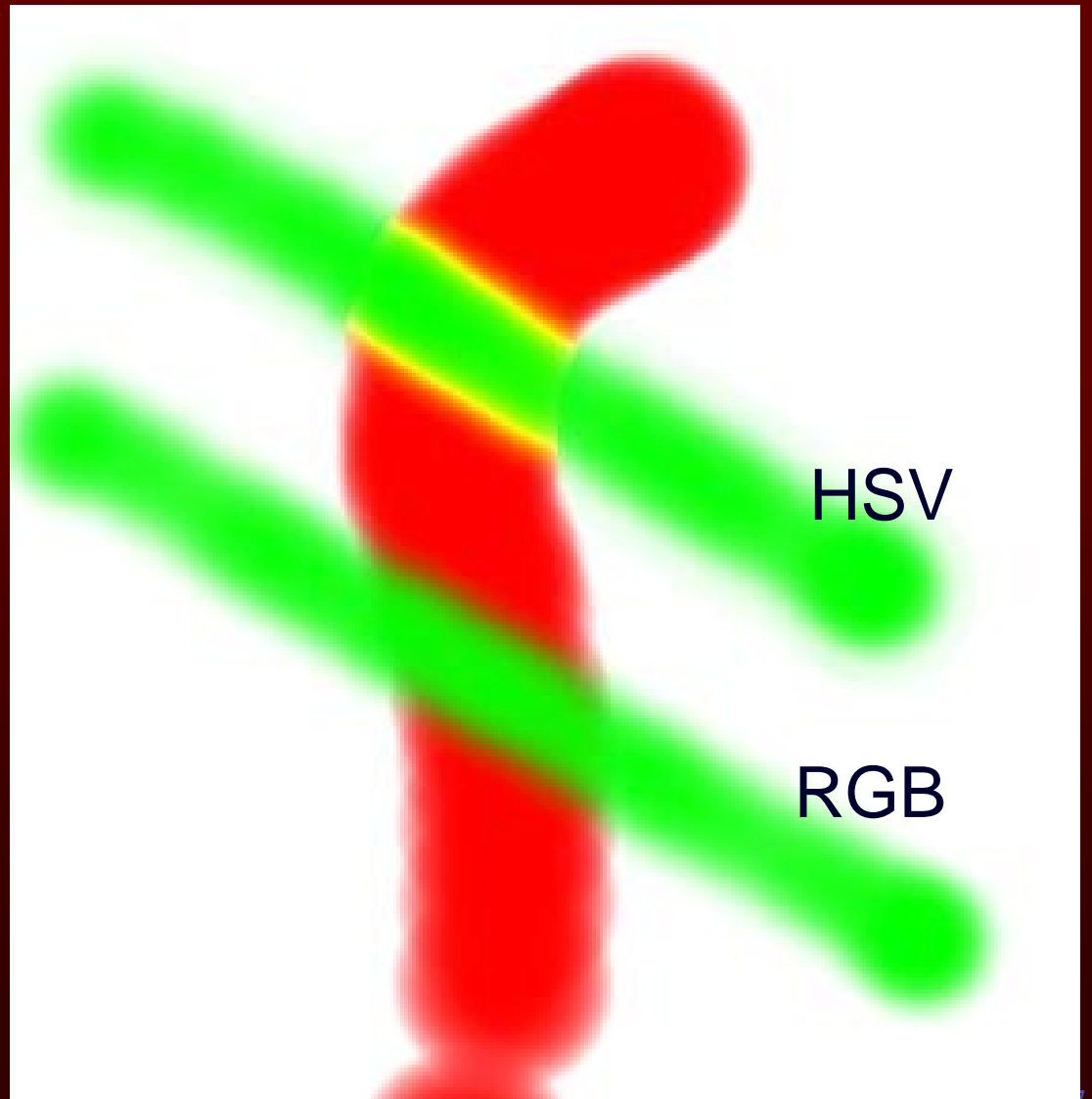


# Part 2: RGB vs. HSV interpolation



Saturation

Hue



## Part 2: Math Example



- Interpolating half way between Red and Cyan ( $\alpha = 0.5$ )
- $\text{NewColor} = 0.5 \text{ Cyan} + 0.5 \text{ Red}$

	R	G	B	H	S	V
Cyan	0.0	1.0	1.0	180	1.0	1.0
Red	1.0	0.0	0.0	0	1.0	1.0
Interpolation	0.5	0.5	0.5	90	1.0	1.0

50% Gray

Greenish

## Part 2: HSV Checkboxes



- Choose which HSV components to affect.
- Allow for any combination.

Interpolate:  H  S  V

```
if (H_check) { NewH = (1- $\alpha$ ) CH +  $\alpha$  PaintH }  
if (S_check) { NewS = (1- $\alpha$ ) CS +  $\alpha$  PaintS }  
if (V_check) { NewV = (1- $\alpha$ ) CV +  $\alpha$  PaintV }
```

# Part 2: Sample Images



Overpainting #1 vs.  
Weighted Mask  
driven painting #2



Image showing H, S, and V Tinting

# Part 3: Brush Visualization



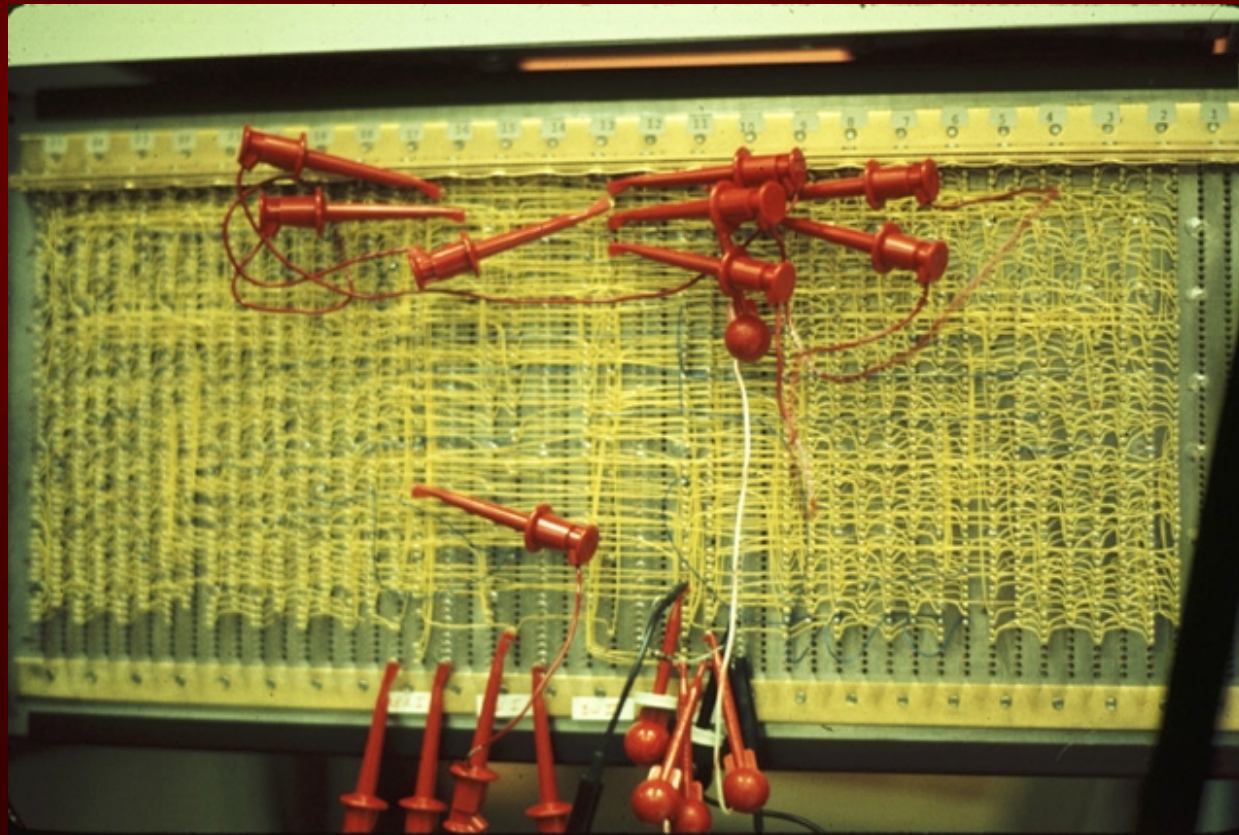
- **Brush Visualization should tell user what its color, falloff and size is**
  - Brush should always be visible regardless of color
  - Draw 1x (actual size) and 4x (four times larger in x and y) versions of the brush
  - Make the larger version discretized – that is it should be a choppy/chunky/pixel replicated version of the actual brush (think xmag, snoop)
  - Make sure this visualization will help you explain to user, TAs, Professor and yourself how the brush weights affect drawing

# Requirements



- **Correctness (40%)**
  - Don't crash
  - Implement all required features
    - (Read the directions like a lawyer)
- **Efficiency (20 %)**
  - No noticeable lag while using your application
- **User Interface (20%)**
- **Programming Style (20%)**
  - Copying code (Don't do it)
- **Submitting with `'/usr/class/cs248/bin/submit'`**

# Paint Program 1973

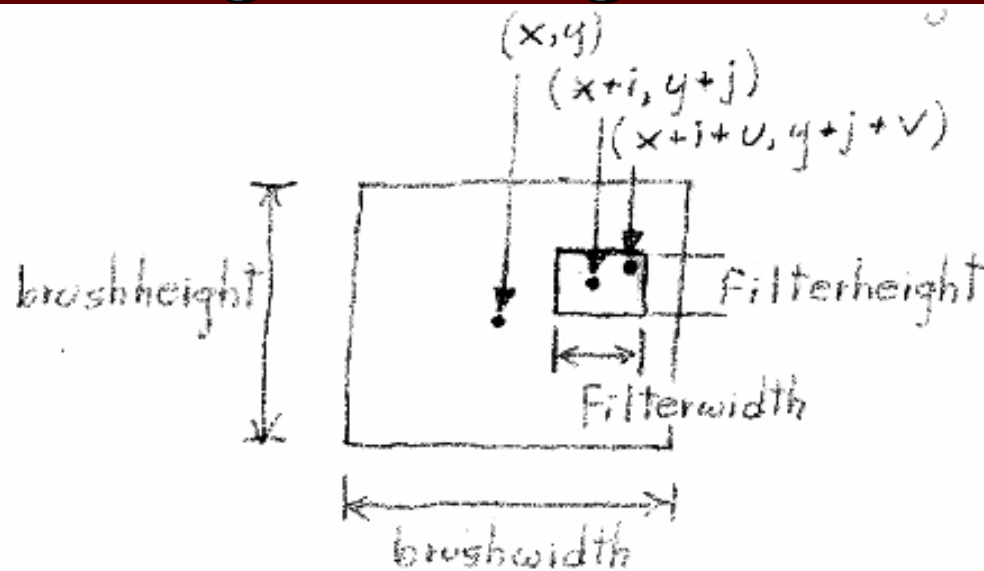


Source: Dick Shoup "SuperPaint: An Early Frame Buffer Graphics System" IEEE Annals of the History of Computing, Vol 23, No 2, Apr-Jun 2001

# Extra credit example



## Blurring the image under the brush.



and  $\sum \omega = \sum_{u,v \in \text{Filter}} \omega(u, v)$

Typical filters are:

1	1	1
1	1	1
1	1	1

or

0	-1	0
-1	4	-1
0	-1	0



# Extra credit inspiration



- Demo of Painter's fancy brushes
- Download Trial from [www.procreate.com](http://www.procreate.com)

# Questions?



- Ask now
- Come to Office Hours
- Email: [cs248-tas@graphics.stanford.edu](mailto:cs248-tas@graphics.stanford.edu)
- Remember: Computer Graphics is fun - if you are not having fun ask TAs for help