Pointillist brush interactive script tutorial.

Introduction.

This tutorial will show how to create an interactive brush which makes it easy to get a pointillism effect with any tool in ArtRage.

Pointillism is the effect of building up a painting out of small dabs of paint colour. With ArtRage tools generally that means making each dot with a mouse click. It's possible to use the glitter brush or the bitmap brush to sample from a tracing image as they apply to get a basic pointillism effect. What we're going to do with this tutorial is allow any tool to rapidly apply dabs of colour sampled from a tracing image.

The script Pointillism.arscript included with this tutorial also has a good example of using an ArtRage script InputBox to control the ways the pointillism effect is produced.

Add Pointillist.arscript to your script actions collection.

Instead of typing the entire script in by hand, we can just copy the included Pointillist.arscript to our Actions folder to have the script appear in our actions. We will go through the script afterwards to show how it works, and then modify the script.

1) From the ArtRage menu bar open the Tools menu then open the Actions submenu. Select the 'Actions Folder…' option. This will open your operating system's file browser to the location where your Actions scripts are stored.
2) Copy the Pointillist.arscript file from this tutorial into the Actions folder.
3) From the ArtRage menu bar open the Tools menu then open the Actions submenu. Select the 'Actions Panel' option to ensure the script Actions panel is open.
4) The Pointillist action should appear in the Actions group. If it isn't there you may need to close and re-open the Actions panel to refresh the list.

Run the Pointillist.arscript

1) Select the ArtRage crayon tool, set to around 50% size. Do a stroke to ensure you're painting in a visible colour and the crayon settings produce something visible.
2) From the Actions panel, run the Pointillist.arscrip action.
3) An input panel should immediately pop up to Set Script Values. Hit 'OK' to use the default options.
4) A notification panel "Waiting for Drag" will appear over the canvas.
5) Do a stroke in the canvas. You should notice the stroke is made up of crayon dots instead of a crayon stroke.

6) Hit ‘Escape’ or click the stop button in the script playback panel to stop the pointillist action.
7) From the ArtRage menu open the Edit menu and select ‘Clear Layer’.
8) From the ArtRage menu open the Tools menu then open the Tracing Options submenu. Select ‘Load Tracing Image…’
9) Run the Pointillist action again.
10) This time in the ‘Set Script Values’ panel try using these input values:
    a. “How much random placement? (0 – 10):” Enter 3
    b. “Rate of dabs (0 – 10):” Enter 10
    c. “Only dab while moving:” Uncheck this option.

    This will give a small amount of random placement, putting the crayon dabs down as fast as possible with no delay, and it will continue to add dabs even if the crayon tool is stationary – similar to how the airbrush tool can do continuous buildup of paint.

11) Hit ‘OK’, and the “Waiting for Drag” notification should pop up over the canvas.
12) Draw around in the canvas over your tracing image. As you draw the crayon dabs should sample the colour from the tracing image and quickly build up a pointillist representation of your tracing image. You can change the size of your crayon tool to add finer detail, or make it bigger for a more abstract effect.

Lets examine the Pointillist.arscript code to see how it works.

Here’s the entirety of the Pointillist.arscript ArtRage scripting code, copied directly from the script file:

```
//============================================================================
// Pointillist brush.
// Copyright Ambient Design Ltd 2019
//============================================================================

<Events>

SetDelay(1)
real rJitter = 0
flag fWaitForMove = true
real rDabRate = 9
real pi = 3.141592653589

InputBox("How much random placement? (0 - 10):  $$rJitter\nRate of dabs (0 - 10):
$$rDabRate\nOnly dab while moving: $$fWaitForMove")
if (rJitter < 0) {
	rJitter = 0
}
else if (rJitter > 10) {
	rJitter = 10;
```

```
}
if (rDabRate < 0) {
	rDabRate = 0
}
else if (rDabRate > 10) {
	rDabRate = 10;
}

while (true) {
	WaitSampleMouse()

	while (MouseButtonIsDown()) {

		if (fWaitForMove) { WaitSampleMouse() }
		else { SampleMouse() }

		real rToolSize = CurrentToolSetting(0x0B2D05E64);
		int nToolID = CurrentToolID()

		real rJitterDistance = 30 * rJitter * rToolSize
		if (nToolID == 4901) rJitterDistance /= 12
		rJitterDistance = Random(rJitterDistance)

		real rDirection = Random(pi * 2)
		real rNewX = MouseX() + sin(rDirection) * rJitterDistance
		real rNewY = MouseY() + cos(rDirection) * rJitterDistance

		// Stroke point.
		<StrokeEvent>
		Loc: (rNewX , rNewY)  Pr: MousePressure()   Ti: MouseTilt()       Ro:
MouseOrientation()    Rv: NO  Iv: MouseInverted()
		</StrokeEvent>

		if (rDabRate == 0) {         // Single Dab only
			while (MouseButtonIsDown()) { }
			break;
		}
		if (rDabRate < 10) {         // Timed Dabs
			real rWaitTime = (0.005 + (1.0 - (rDabRate / 9)) * 0.3)
			Wait: (rWaitTime)
		}
		// else maximum dab rate with no delay.
	}
}
```

Let's examine each line of the script to see how it works:

1) The first few lines starting with '//' are ignored by the script – they're just comments there for our benefit.
2) <Events>; tells the ArtRage script this is where the important stuff starts.
3) SetDelay(1); Normally Action scripts play back as fast as ArtRage can process and render the strokes. Later in the script we want to affect the rate at which dabs can be applied so we tell the script to play back at 1x the normal time periods. If we'd SetDelay(0) instead, we wouldn't be able to use the 'Wait:' directive later in the script to slow the dabs down.
4) real rJitter = 0; This will be the value of 0 – 10 the amount to offset the dabs from the mouse point for random placement
5) flag fWaitForMove = true; This flag says whether the dabs will be applied continuously, or only when the mouse is moving.
6) real rDabRate = 9; This is the rate dabs will be applied. 0 – 10 where 10 is as fast as possible, and 0 is only one dab each time we click.

7) real pi = 3.141592653589; We need PI for trigonometry maths later.
8) InputBox("How much random placement? (0 - 10): $$rJitter\nRate of dabs (0 - 10): $$rDabRate\nOnly dab while moving: $$fWaitForMove"); This brings up the 'Set Script Values' input box. There are several important parts to this line so I'll break it up into parts:
    a. InputBox("How much random placement? (0 - 10):
       This creates the input box and puts the first text on the first line
    b. $$rJitter
       This tells the input box we want to use the variable 'rJitter' as input for a value. Because rJitter is a 'real' variable type the input box creates a text entry box where we can type in a number. That number will be given to the rJitter variable when the input box 'OK' button is pressed later.
    c. \n
       This tells the input box that any more text will go onto a new line.
    d. Rate of dabs (0 - 10): $$rDabRate\n
       Similarly this adds the text for the dab rate, creates the number input for the rDabRate variable, then tells the input box to get ready for another new line.
    e. Only dab while moving: $$fWaitForMove
       Similarly this adds the text for the wait for movement option. This is slightly different because the variable 'fWaitForMove' is a 'flag' type variable. So the input box knows to put a checkbox which will make fWaitForMove 'true' or 'false' when 'OK' is pressed later.
    f. ")
       The quote mark closes off the string, and the bracket closes off the function call.

   When you hit 'OK' the number entered in the first box is assigned to 'rJitter', the number typed in the second box is assigned to 'rDabRate', and the checkbox sets the 'fWaitForMove' value. If you'd hit 'Cancel' instead the script ends immediately.
9) The next several lines ensure that the values entered are inside the valid ranges we can use. It's always good practice to check entered values are within the ranges you expect.
    a. If (rJitter < 0) {
           rJitter = 0
       }
       If the number entered into the first box was a negative number (a number less than zero) then rJitter will be set to zero – the minimum value we allow.
    b. else if (rJitter > 10) {
           rJitter = 10;
       }
       Otherwise if the number entered into the first box was greater than 10 we'll set rJitter to 10. All other numbers between zero and ten are acceptable so we don't need to change rJitter
    c. if (rDabRate < 0) {
           rDabRate = 0
       }
       else if (rDabRate > 10) {
           rDabRate = 10;
       }
       Similarly if the number entered in the second box was less than zero or greater than ten rDabRate is clamped.

10) while (true) { This is the start of our 'outer loop'. In a 'while' function we do the code inside the '{' and '}' block until the value in brackers '(' and ')' after the while statement is true. In this case (true) can never be false – it's always true. So we're saying to loop forever! The only way to stop this script is to hit the escape key, or hit the stop button in the script playback panel. This is desirable because it means the script allows the user to do many strokes in the canvas without having to restart the script for every stroke.
11) WaitSampleMouse(); Tell the script engine to pause until the user clicks or does a stroke in the canvas. This brings up the "Waiting for Drag" notification in the canvas.
12) while (MouseButtonIsDown()) { This is our 'inner loop'. So long as the mouse button is down, this block of script code will repeat. We will keep doing dabs of paint so long as the mouse button is held down.
13) if (fWaitForMove) { WaitSampleMouse() } If the user checked the 'Only dab while moving' option in the input box, fWaitForMove will be true. If so we pause playback of the script here until the mouse moves.
14) else { SampleMouse() } Otherwise, if fWaitForMove is false (the user turned off the checkbox in the input box) we will just sample the current mouse location (and pressure and other values) but not pause the script. If the mouse hasn't moved the dab will happen in the same place as the previous time we looped through this bit of script.
15) real rToolSize = CurrentToolSetting(0x0B2D05E64); I'm going to base the distance of the brush jitter off the size of the tool. Larger tools need to displace the dabs further to give the same relative effect as a smaller tool. For example a 100 pixel size brush has to offset the dab 100 pixels to be offset one dab-size from the mouse point. A 20 pixel brush needs to offset 20 pixels to be one dab-size from the mouse point. CurrentToolSetting gets the setting for whatever tool is the currently active tool, where the ID of the setting matches what is in the brackets. So the setting with ID '0x0B2D05E64' is the brush size setting. The easiest way to find what ID you want to mess with is to record a script where you change the tool setting, then look in the script for (for example):
```
Wait: 0.000s      EvType: Command      CommandID: SetToolProperty   ParamType: ToolProp   Value: { 0x0B2D05E64 (Size), 0.50 }
```
This is a command event 'SetToolProperty', and the Value: section gives the ID of the property, then the value that property was set to. We can use that ID in the CurrentToolSetting function to get that value from the tool.
16) int nToolID = CurrentToolID(); In ArtRage all the tools are roughly the same size range except the pencil tool. We're going to need to change our dab jitter for the smaller size ranges of the pencil tool. CurrentToolID() returns the ID of the current tool, and we're storing it in the nToolID integer variable for checking later.
17) real rJitterDistance = 30 * rJitter * rToolSize; To calculate the number of pixels to offset the dab from the mouse point, we'll allow rJitter of 0 – 10, rToolSize is 0 – 1, multiplied by 30 to give a value from 0 – 300. As the tool gets bigger the rJitterDistance increases, and we can limit how far using rJitter.
18) if (nToolID == 4901) rJitterDistance /= 12; If the tool ID is the pencil tool we want the distance to be 1/12th so the jitter matches the smaller size of the pencil. Like getting the CurrentToolSetting ID above, we can grab the ID of any tool by recording a script where we select a tool in the tool picker. Then look in the script for a line similar to this:
```
Wait: 0.000s      EvType: Command      CommandID: CID_ToolSelect    ParamType: ToolID     Value: { 4900 (Oil Paint) }
```
This is a command event to select a tool. The Value: section gives the ID of the tool (in this case 4900 for the Oil paintbrush).

19) rJitterDistance = Random(rJitterDistance); We want the actual jitter distance to be a random value between zero and the calculated maximum jitter distance. Random(x) returns a value between zero and x, so Random(rJitterDistance) gives a value between zero and rJitterDistance.
20) real rDirection = Random(pi * 2); This is the circular direction the dab will be offset from the mouse point. The value is in radians, so we get a random value between zero and 2 x PI
21) real rNewX = MouseX() + sin(rDirection) * rJitterDistance;
real rNewY = MouseY() + cos(rDirection) * rJitterDistance;
Trigonometry! You might have vague memories of high-school maths where the sine and cosine of an angle in a right-angle triangle can be used to get the length of the sides adjacent to and opposite from the angle… Well anyway we have the direction (the angle) and we have the distance rJitterDistance (length of the hypotenuse) so we can calculate the other two sides as sin(rDirection) * rJitterDistance. This is the offset from the mouse location to place our dab. So we just simply that to the mouse location for the X and Y values.
22) // Stroke point. I'm a comment. The script ignores me. I'm just here to either make the script more readable for humans, or to confuse them more.
23) <StrokeEvent>; This defines the actual start of a scripted paint stroke.
24) Loc: (rNewX , rNewY) Pr: MousePressure() Ti: MouseTilt() Ro: MouseOrientation() Rv: NO Iv: MouseInverted()
We're only doing a single dab of paint, so there is only one mouse event inside our stroke event. The centre point of our dab is at Loc: (rNewX, rNewY) which we calculated from the mouse location + jitter offset and direction. Then we just pass in the other sampled mouse (or graphics tablet stylus) values of pressure, tilt, rotation.
25) </StrokeEvent>; The end of our stroke event block. There was only one mouse event, so it was a dab of the tool rather than a long paint stroke.
26) if (rDabRate == 0) { If we entered a dab rate of zero in our input box we want to only do a single dab per mouse stroke. In that case this block of code will be run.
    a. while (MouseButtonIsDown()) { } So long as the mouse button is down, do absolutely nothing at all. So we get to here, the dab was previously painted, but we don't let the script go any further while the use keeps the mouse button down. We're stuck at this line and cant do any more dabs until the user lets go of the mouse button (or lifts the stylus)
    b. break; This will make us 'fall out' of our larger 'while (MouseButtonIsDown())' loop we started back at bullet-point 12. But there's no script code to execute at the end of that 'while' block, so we loop all the way back up to 'while (true) {' which is way back at bullet point 10.
27) } Close off the block for single dabs – when rDabRate is zero. To recap… If we have rDabRate of zero we want to allow only one dab per stroke. So when the mouse goes down we paint the single dab, then we stall the script for as long as the mouse stays down. As soon as it goes up we loop all the way back to where we wait for the next time the mouse goes down. So only one dab can ever be drawn per stroke in the canvas.
28) if (rDabRate < 10) { For almost every other value of the rDabRate we want to add in a delay to slow the dabs down. Note that if the rDabRate was zero, we never get to this point in the script, and if the value is 10 we get here, but we don't go into the block where we'll add the delay. So a value of 10 will keep draw paint dabs as fast as possible.

    a. real rWaitTime = (0.005 + (1.0 - (rDabRate / 9)) * 0.3); Maths! rWaitTime is in seconds. The minimum wait time is 0.005 seconds (5 milliseconds). rDabRate is a number between 1 and 9, so we divide by 9 to get a number between 0.111 and 1. If we have a larger rDabRate, we want a smaller delay, so subtracting the dab rate from 1 gives us a delay amount. And we want the delay to be between 0.033 (33 milliseconds) and 0.3 seconds (300 milliseconds) so we multiply the result by 0.3 seconds, and add it to our minimum wait time. So we've converted our rDabRate of 1 – 9 to a delay between 38 milliseconds to 300 milliseconds. Or something like between 30 dabs per second down to 3 dabs per second. Maths.
    b. Wait: (rWaitTime); Pause script execution for rWaitTime seconds.

29) } Close off the block for timed dabs – for rDabRate between 1 and 9. To recap… If rDabRate was 0 we never got to this block at all. If rDabRate was 10 we got to the block but we didn't go into it, so no delay was added. If rDabRate was between 1 and 9 we entered the block and paused the script for a short while related to the dab rate – longer pause for a lower dab rate, shorter pause for a higher rate of adding dabs.
30) // else maximum dab rate with no delay. Just clarifying to myself that rDabRate of 10 didn't do either of the 'if' cases above so there was no delay and dabs happen continuously.
31) } This is the end of our while (MouseButtonIsDown()) { block from bullet point 12 above. All the code inside the '{' and '}' pair which define our mouse stroke block will keep looping until the user ends the stroke in the canvas.
32) } This is the end of our while (true) { block. When we hit this point we loop back to bullet point 10 and test to see if 'true' is still true. True is always true so we loop back into our while (true) { block, where we will immediately wait for the user to do another stroke in the canvas. Forever. There's no escape. Actually, there is escape when you hit the escape key, or click the stop button in the script playback panel to end our pointillist.arscript.

Conclusion.

To recap… We created an input box to get the amount of random placement (rJitter), how fast we place the tool dabs (rDabRate), and whether the dabs happen continuously or only when the mouse is moving (fWaitForMove).

We then clamped the input values to make sure they are values we can use.

Then we started the forever loop of waiting for the user to do strokes in the canvas.

We either wait for the mouse to move before getting the mouse values, or we just grab the current mouse values, depending on whether the 'Only dab while moving' (rWaitForMove) option was selected.

And when they start a stroke we work out from the tool size and whether it's a pencil, based on the amount of jitter how far to offset the paint dab. And we give a random offset from the mouse point.

We do our single dab of the tool.

Now we stall the script for a single dab until the user releases the mouse or lifts their graphics tablet stylus. Or we stall a small amount to limit the speed dabs are laid down. Or we don't stall at all for the highest rate of dab placement.

When the user ends their stoke we go back to our forever loop so the user can start a new stroke.

Eventually the escape key or stop button from the script playback panel will end our script.

Whew.