

Vine Brush Interactive Script Tutorial.

Introduction.

This tutorial demonstrates how to draw a single leaf, then use that leaf to make a brush that adds leaves as you draw a vine. This is an advanced tutorial for people who are familiar with ArtRage script action recording, script programming, and who can edit text files using a basic text editor. On Windows we can use NotePad, and on OSX we can use TextEdit. (If you want to use a different text editor you may need to be aware ArtRage script files are required to be in UTF-16 text format.)

Draw a basic leaf.

The included script 'leaf.arscript' shows the process for drawing a single leaf.

1. Select the pencil tool, around 50% size, non-precise.
2. From the ArtRage menu, open the Tools menu, open the Actions submenu, and select the Actions panel to ensure the Actions panel is open.
3. Click the 'Record Action' button to record a new script action.
4. In the Record Action panel name your action 'MyLeaf'.
5. Ensure that **both** the 'Record on New Layer' and 'Record Tool Settings' are turned **off**. We only want to record the process of drawing the leaf, without any other events.
6. Click 'Ok' to begin recording.
7. Draw a single leaf, about 140 pixels in height. The leaf should be oriented so the stem of the leaf points downward and the tip of the leaf is upwards.
 - a. Draw the stem first, starting from the base where it would join the vine. This makes it easy to work out the mouse coordinates for the base of the leaf, which we want to connect to the vine. The coordinates will be the first stroke point in the script.
 - b. We want this script to be able to be used with any tool at any size, so avoid making any changes to tool settings. Just keep using the pencil with its current settings.
 - c. Draw the outline and veins of the leaf.
 - d. Click the Stop button in the script Recorder panel to save the action.
8. Play the action 'MyLeaf' from the action panel to ensure it works correctly, before we start messing with it.



Edit the leaf action script to make the vine brush.

Before we start editing our leaf action script, we should make a copy.

1. In the Actions panel right-click on the MyLeaf action and select 'Duplicate Item...'

2. Right click on the copy 'MyLeaf Copy' and the select 'Rename Item...' menu option.
3. Enter the name 'MyVine'. We will edit the new MyVine action script to allow the pencil tool to be used to add leaves to a vine as we draw.

Locate the saved action script and open for editing

We're going to edit our saved action script with a text editor – probably NotePad on Windows, or TextEdit on OSX.

- 1) From the ArtRage menu, open the 'Tools' menu, and open the 'Actions' submenu. Select the 'Actions Folder...' option. This should open your operating system's file browser.
- 2) Using NotePad on Windows, or TextEdit on OSX open the file MyVine.arscript for editing.
- 3) The first lines of your script should look similar to this:

```
//=====
//=====
//                               ArtRage Script File.
//=====
//=====

//=====
// Version Block - Script version and ArtRage version:
//=====

<Version>
  ArtRage Version: ArtRage 3 6
  ArtRage Build: 6.0.0
  Professional Edition: Yes
  Script Version: 1
</Version>

//=====
// Header block - Info about the painting/person who generated this script:
//=====

<Header>
  // === Project data
  Painting Name: "Untitled"
  Painting Width: 1280
  Painting Height: 994
  Painting DPI: 72
  Mask Edge Map Width: 1280
  Mask Edge Map Height: 1024
  // === Author data
  Author Name: ""
  Script Name: "MyLeaf"
  Comment: ""
  Script Type: ""
  Script Feature Flags: 0x00000000
</Header>

//=====
// Script data follows:
//=====

<Events>
```

And then following the <Events> tag should be the actual recorded events for the painted leaf. There should be several <StrokeEvent> blocks, and loads of lines which start with 'Wait: 0.019s' (or

similar). Because this is a copy of our action script for painting a single leaf, you might notice 'Script Name: "MyLeaf"' in the <Header> block. This is fine – you can ignore that (or change that name to "MyVine" if you like)

Change the script to draw the vine

This is the advanced bit. We going to change the script to interact with the mouse, draw using the current tool, and every so often draw a vine leaf. As the vine advances we want the leaves to appear on both sides of the vine, and to follow the direction of the drawn line.

To do this we will make the original leaf into an ArtRage script function we can call many times, and give it some parameters for how the leaf is to be drawn. Then we'll need to write some ArtRage script code to interact with the mouse, calling the leaf function as we go.

Change the script to draw a single leaf with the current tool and colours.

The leaf should be able to be drawn with whatever current tool and colour are selected. We need to make sure our leaf script was recorded without explicitly setting the tool and colour settings.

- 1) Edit your MyVine.arscript file with your text editor. We want to make the leaf function as simple as possible and to use whatever tools, colours and settings are active before it was called.

- a. After the <Events> tag there may be a number of lines of script which look similar to this:

```
Wait: 0.000s    EvType: Command    CommandID: Add New Layer
Wait: 0.000s    EvType: Command    CommandID: ToolPreset ToolID: 4906
(Eraser)        Tool Data: {

    [Timeline diagram showing a sequence of tool preset events for the Eraser tool.]

} // End of tool preset binary data.
Wait: 0.000s    EvType: Command    CommandID: ToolPreset ToolID: 4901
(Pencil)        Tool Data: {

    [Timeline diagram showing a sequence of tool preset events for the Pencil tool.]

} // End of tool preset binary data.
Wait: 1.092s    EvType: Command    CommandID: SetForeColour    ParamType:
Pixel Value: { 0xFF8052E9 }
```

- b. Those are commands which create the new layer, select the tool including its settings, set up the current colour, and possibly other events depending on what actions you took after starting up the script recording. Ideally we don't want any of those commands – we just want the stroke events using the tool settings active.
 - c. Delete everything **between** the <Events> tag and the first <StrokeEvent> tag, but **do not** delete the <Events> and <StrokeEvent> tag.
 - d. You might find other command ID events between the end of one stroke block and the start of another if you made tool changes while you were recording the original leaf script. Ideally we want to delete all of those as well. So every </StrokeEvent> for the end of one stroke should immediately be followed by a new <StrokeEvent> tag on the next line, except for the very last </StrokeEvent> at the end of the script.
- 2) Save your MyVine.arscript file and run the action in ArtRage (You might need to close and re-open the Actions panel if you've copied or added any other files with the file browser).

You should now be able to select any tool and any colour, and when you play the MyVine action the leaf should be drawn in the current tool and current colour.

Change the script so the leaf drawing happens inside a function we can call.

Now that the leaf can be drawn in any colour with any tool, we can make it a function which we can call repeatedly. We're going to want to change the position of our leaf, as well as drawing the leaf on opposite sides of the vine as we go. We might also want to let the leaf drawing use stylus pressure to match the vine. The <Events> tag defines the starting point of ArtRage script events. It doesn't matter if functions appear before or after the <Events> tag. For ease of explanation I'm putting the function definitions after the <Events> tag, but the function definitions are ignored until the first bit of code outside a function definition is found.

- 1) **After** the <Events> tag, but **before** the first <StrokeEvent> we want to put the function definition for the single leaf:
`void Leaf(real x, real y, real rRotation, real rScale, real rPressure) {`
- 2) **After** the last </StrokeEvent> add another line with a close brace:
`}`
- 3) Make sure there's a blank, empty line at the end of the script – it ensures the last command was fully read in and completed.
- 4) This has made all the leaf drawing happen inside a function called 'Leaf', and we can pass in a location, rotation, scale, and pressure real number values. If we were to run the script now, nothing would happen because the function is never called, so the leaf drawing strokes never happen.
- 5) Don't panic – it is supposed to be broken at this point.

Use the function parameters to change how the leaf is drawn.

We're passing in location and orientation parameters for the leaf – we need to use them when the leaf is drawn. We **could** edit every single line of the script to add the location and rotation and pressure information to every single stroke point in the strokes that make up the leaf, but that would take a long time and we'd almost certainly get it wrong. It is much simpler to change the space in which the leaf is drawn to match how we'd like the leaf to appear. We can add a transformation offset so the stroke events happen in a different place than where they were recorded. We can add a transformation rotation so the leaf is drawn at a different angle. And we can add a pressure transformation so the leaf is drawn with the new stylus pressure. Unfortunately if we change this transformation 'space', we need to change it back to how it was before we changed it. Otherwise anything that isn't the leaf will also happen in the new canvas transformation space. So we're going to 'push' the original space onto the transformation stack, change the space, and when we've finished drawing the leaf, 'pop' the original space back off the stack.

- 1) Inside our Leaf function the first thing to do is preserve the old transformation space before we mess with it. So **after** the function definition add the following line:
`PushState()`
This puts the settings for the rotation, scale, and offset of the current drawing space onto a 'stack'. It also adds the current colour settings. Now if we change any of those settings we can get back to the original settings by 'popping' the old settings off our stack.
- 2) It is **very important** that every time we save the transform state on the stack we take it back off the stack later. Otherwise our stack will use up all the memory. So at the end of our Leaf function, **before** the last close brace '}' we want to add the following line:
`PopState()`

- 3) Now anything we do to transform space between the PushState and PopState won't affect anything outside the function. Note that the transform stack also preserves the current colour, so if you happen to want to make the leaves particularly beautiful you could change the tool colour when you originally recorded the leaf script.
- 4) We want to make all the transformations of our leaf based around where the stem of the leaf would connect to the vine as we draw. Fortunately when we drew our leaf we started at the base of the stem, so the very first stroke point tells us where the base of the stem for the leaf is. Note that the number in the 'Loc:' section will be different from the values I give below. The first part of the stroke data should be similar to this:

```
<StrokeEvent>
  <StrokeHeader>
    <EventPt>      Wait: 0.000s   Loc: (499, 293)      Pr: 1   Ti: 1   Ro:
0      Fw: 1   Bt: 0   Rv: NO   Iv: NO </EventPt>
```

- 5) If you have used a graphics tablet and stylus the number values may include decimal places and may look more like this:

```
<StrokeEvent>
  <StrokeHeader>
    <EventPt>      Wait: 0.000s   Loc: (466.667, 464.333)      Pr:
0.626223      Ti: 0.622222   Ro: 0.238889   Fw: 1   Bt: 0.025   Rv: NO   Iv: NO
</EventPt>
```

- 6) In either case the two numbers we want are the X and Y coordinates of the first event point, which are inside the brackets after the 'Loc:' parameter. In the first case it would be 499, 293 and in the stylus case it would be 466.667, 464.333. Remember, your numbers will be different to mine.
- 7) We want to move the transformation offset so the leaf will be drawn at the mouse coordinates we pass into the function. **After** the PushState() line in our function add the following two lines:


```
SetTransOffset(x - 466.667, y - 464.33)
SetTransCentre(466.667, 464.333)
```
- 8) **Important note:** The numbers you should use in your SetTransOffset and SetTransCentre functions should be the two values from your Loc: parameter. They will be different to the ones above.
- 9) The first line – SetTransOffset(x - 466.667, y - 464.33) – subtracts the original leaf coordinates (so the leaf would be drawn at (0, 0)) and adds in the mouse coordinates we passed into the function (so the leaf is now drawn at (x, y))
- 10) The second line - SetTransCentre(466.667, 464.333) – sets the centre point for the next transformations we're going to do. We have the leaf starting point at the base of the stem, and we want that point to stay in the same place if we scale and rotate the leaf, so the leaf rotates around the base of the stem.
- 11) After the SetTransCentre line, add the following three lines:


```
SetTransScale(rScale, rScale)
SetTransRotation(rRotation)
SetTransPressure(rPressure)
```
- 12) Those three functions take the passed-in parameters from our Leaf function and change the scale the leaf is drawn at, the rotation of the leaf, and the amount of pressure.
- 13) Our Leaf function should look something like the following. (Be aware your location values will be different, and I've removed the stroke events for brevity):


```
void LeafA(real x, real y, real rRotation, real rScale, real rPressure) {
  PushState()
  SetTransOffset(x - 466.667, y - 464.33)
  SetTransCentre(466.667, 464.333)
  SetTransScale(rScale, rScale)
```

```

SetTransRotation(rRotation)
SetTransPressure(rPressure)
<StrokeEvent>
    ... Stroke events removed for brevity. There will be a whole bunch of them in
    here...
</StrokeEvent>
PopState()
}

```

14) Save your MyVine.arscript

15) At this point it still doesn't do anything – we're still not calling the function. So you don't need to panic just yet. You **should** double-check you have PushState and PopState functions inside your Leaf function, that you have an open-brace '{' at the start of your function, and a close-brace '}' at the end of your function. And that the values you used in the SetTransOffset and SetTransCentre functions are the ones from **your** Loc: parameters and not mine.

Can we make it do something now?

Now would be a good time to test that our function works. We're going to add some ArtRage script code to pause until the mouse is clicked in the canvas, then draw a spiral of leaves at the mouse click. The leaves will be drawn with the current tool settings in the current colour.

1) At the very end of our script, **after** the last close-brace '}' which closes off our Leaf function, add the following lines:

```

SetDelay(0)
WaitSampleMouse()
real pi = 3.14159265
real rUpX = MouseX()
real rUpY = MouseY()

int nTotal = 10
real rSmall = 0.2
real rLarge = 1
int n
for (n = 0; n < nTotal; n++) {
    real rLeafRotation = 1.0 / nTotal * n * pi * 2
    real rLeafScale = rSmall + ((rLarge - rSmall) / nTotal * n)
    real rPressure = 1
    Leaf(rUpX, rUpY, rLeafRotation, rLeafScale, rPressure)
}

```

2) The code lines do the following:

- SetDelay(0) - sets playback to be at maximum speed – we want the leaves to be drawn as fast as possible.
- WaitSampleMouse() - pauses script playback, waiting for the user to click in the canvas. ArtRage will bring up a notification that it is waiting for a mouse-click in the canvas. It also samples the mouse coordinates for use with MouseX() and MouseY() later.
- real pi = 3.14159265 – stores the value for pi in a variable called pi. Useful for doing trigonometry.
- real rUpX = MouseX() and real rUpY = MouseY() – gets the mouse coordinates and stores them for later.
- int nTotal = 10 – this is the number of leaves we're going to draw.
- real rSmall = 0.2 – this is the scale of the smallest leaf

- `real rLarge = 1` – this is the scale of the largest leaf.
 - `int n` – a general variable we'll use as our loop counter.
 - `for (n = 0; n < nTotal; n++) {` This is the start of our loop block. We'll loop 'nTotal' number of times, and each time 'n' will be incremented by 1
 - `real rLeafRotation = 1.0 / nTotal * n * pi * 2;` There's that trigonometry. We divide our circle into 'nTotal' pieces and use the 'nth' piece. Rotation is done in radians, where a full circle is $2 \times \pi$ radians so we multiply by $\pi * 2$
 - `real rLeafScale = rSmall + ((rLarge - rSmall) / nTotal * n);` The scale of the leaf is the smallest leaf plus the difference between the largest and smallest multiplied by which leaf we're working on.
 - `real rPressure = 1;` We're all under a lot of pressure.
 - `Leaf(rUpX, rUpY, rLeafRotation, rLeafScale, rPressure);` Call our single leaf function to draw the leaf at the mouse coordinates, rotated and scaled depending on how far through the loop we are.
 - `}` closes off our loop block. All the script lines between the '{' and '}' in our loop block will be processed nTotal number of times.
 - Remember to have a blank line after the last script line.
- 3) Save MyVine.arscript and try running it from the ArtRage actions panel. All going well you should get a number of leaves radiating from the mouse-down point in the canvas, getting larger as they go. They will use the current tool and current colour so you can select different tools and colours



- 4) If you get errors, or you don't get something like the above drawn at your mouse click, take a look at Leaves.arscript in this tutorial. Your lines of ArtRage script code should be the same as the code at the bottom. Feel free to cut'n'paste from the end of Leaves.arscript to see if it works. If it still won't do anything useful there's probably a mistake at the start of your 'void Leaf(real x, real y, real rRotation, real rScale, real rPressure)' function. Check those lines of ArtRage script code as well (or copy them from Leaves.arscript).
- 5) You might want to save this script under a different name so you can play with it more later.

So how about drawing the vine.

The next step is to write a block of script code to track the mouse as we draw in the canvas and draw the vine as we go. As the mouse is dragged in the canvas we will draw a segment of vine. When the segment of vine is long enough we will draw a leaf on either one side or the other side of the vine. And when the mouse goes up we will stop drawing the vine.

We will need two loops: The inner loop tracks the mouse movement and draws the bits of vine until we've drawn enough vine to need to draw a leaf. The outer loop calls the inner loop to draw a segment of vine, then when the segment is done the outer loop draws a single leaf. When the mouse button is released the outer loop stops and the script ends. In pseudo-code it would look something like this:

```

Wait for the user to start the vine.
While the mouse is down {
    Get the mouse coordinates
    Draw the start point of a stroke. <StrokeEvent> and StrokeHeader
    While we haven't travelled far enough for a segment of vine {
        Get the mouse coordinates
        Draw a small piece of a paint tool stroke
        Measure how far along the segment of vine we've travelled
    }
    Finish off the tool stroke for this segment of vine. </StrokeEvent>
    Draw a single leaf. Each alternate leaf is drawn on the opposite side of the vine.
}

```

If you're working with your script which drew the ten leaves you'll need to delete all the lines of ArtRage script code after the end of the Leaf function – everything from the SetDelay(0) line onwards. We want to add new script code after the '}' that defines the end of the Leaf function. We're going to add three new blocks of ArtRage script code: two utility functions and then the vine drawing code that calls all our functions.

- 1) We need two utility functions to help with the vine drawing. Here's the first


```

real VectorLength(real x, real y)
{
    return sqrt(x * x + y * y)
}

```

If you remember your high-school geometry, the hypotenuse of a right-angle triangle is the square of the sum of the square of the other two sides. Or $a^2 + b^2 = c^2$ Or to put it another way: The distance between two points is the square-root of the difference in their X and Y coordinates squared and added.

- 2) The next utility function helps the script engine set up the start of a stroke. When a stroke of paint starts ArtRage needs to know information about what direction the stroke is intended to travel for setting head angles and smoothing. Inside every <StrokeEvent> block there's a <StrokeHeader> block which records that information when we record a script. If we're generating an entirely new stroke we need to set up that stroke header block. Copy the following code and add it to the end of your script.

```

void StrokeHeader(real x, real y, real rOldX, real rOldY, real p, real t, real r,
flag i)
{
    real rDX = x - rOldX;
    real rDY = y - rOldY;
    real rVLen = VectorLength(rDX, rDY);
    if (rVLen == 0) rVLen = 1;
    real rNX = rDX / rVLen;
    real rNY = rDY / rVLen;
    // Do the stroke header - sets up head angle and anticipated stroke
    direction.
}

```



```

        <StrokeHeader>
            <EventPt>      Wait: 0.000s   Loc: (x, y)   Pr: p   Ti: t   Ro: r   Rv:
NO      Iv: i  </EventPt>
            <Recorded>    No      </Recorded>
            <Smooth>      Count: 3
                        Loc: (rOldX, rOldX)   Pr: p   Ti: t   Ro: r
                        Loc: (x - rDX / 3 * 2, y - rDY / 3 * 2)   Pr: p   Ti:
t      Ro: r
                        Loc: (x - rDX / 3,      y - rDY / 3)   Pr: p   Ti:
t      Ro: r
            </Smooth>
            <PrevA>Loc: (rOldX, rOldX)   Pr:
p      Ti: t   Ro: r  </PrevA>
            <PrevB>Loc: (x - rDX / 2,      y - rDY / 2)   Pr: p   Ti: t   Ro:
r      </PrevB>
            <OldHd>Loc: (rOldX, rOldX)   Pr:
p      Ti: t   Ro: r   Dr: (rNX, rNY) Hd: (-rNY, rNX)   </OldHd>
            <NewHd>Loc: (x - rDX / 2,      y - rDY / 2)   Pr: p   Ti: t   Ro:
r      Dr: (rNX, rNY) Hd: (-rNY, rNX)   </NewHd>
        </StrokeHeader>
    }

```

This function takes all the stylus information like location, pressure, tilt, rotation, barrel twist and produces a stroke header block based on the direction of travel of the stylus. This is a useful function to have if you're creating ArtRage tool strokes in the canvas using ArtRage script language and want the stroke header to be set up correctly. It isn't necessary to examine every line, but in general terms it compares the previous (old) mouse coordinates to the new ones and sets up the brush head direction and start of stroke smoothing. We will call this function to start off every segment of the vine to give a clean stroke start.

- 3) What follows is a great wedge of ArtRage script code to track the mouse and draw the vine segments and the leaves. Copy the great wedge of code and paste it at the end of your script. Explanation of the code will follow:

```

SetDelay(0)
WaitSampleMouse()
real rLastX = MouseX()
real rLastY = MouseY()
real rStrokeEndX = rLastX
real rStrokeEndY = rLastY

real rSegLen = 30
real pi = 3.141592653589
flag fTopLeaf = true
real rDirectionX = 0
real rDirectionY = 0
real rDX = 0
real rDY = 0

// Full vine loop
while (MouseButtonIsDown()) {

    // Start vine segment
    <StrokeEvent>
        StrokeHeader(rStrokeEndX, rStrokeEndY, rLastX, rLastY, MousePressure(),
MouseTilt(), MouseOrientation(), MouseInverted())

    // Vine segment loop

```

```

real rStrokeLen = 0
while (MouseButtonIsDown() && rStrokeLen < rSegLen) {
    WaitSampleMouse();
    real rNewX = MouseX()
    real rNewY = MouseY()

    rDX = rNewX - rLastX
    rDY = rNewY - rLastY
    real rVLen = VectorLength(rDX, rDY)

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

    rLastX = rNewX; rLastY = rNewY
    rStrokeLen += rVLen
}

// If we started the stroke, we must end the stroke.
rStrokeEndX = MouseX()
rStrokeEndY = MouseY()

// Do two stroke points to finish off. One for the last mouse draw location,
and one for mouse up location.
Loc: (rStrokeEndX, rStrokeEndY) Pr: MousePressure()      Ti: MouseTilt()
Ro: MouseOrientation()      Rv: NO Iv: MouseInverted()
Loc: (rStrokeEndX, rStrokeEndY) Pr: MousePressure()      Ti: MouseTilt()
Ro: MouseOrientation()      Rv: NO Iv: MouseInverted()

</StrokeEvent>

// Draw a leaf
real rLeafScale = 0.5
real rLeafRotation = 0.5
if (fTopLeaf) rLeafRotation = pi - rLeafRotation
fTopLeaf = !fTopLeaf
rDirectionX = rDirectionX * 0.1 + rDX
rDirectionY = rDirectionY * 0.1 + rDY
rLeafRotation -= atan2(rDirectionX, rDirectionY) + 3 * pi / 2

Leaf(rStrokeEndX, rStrokeEndY, rLeafRotation, rLeafScale, MousePressure())
}

```

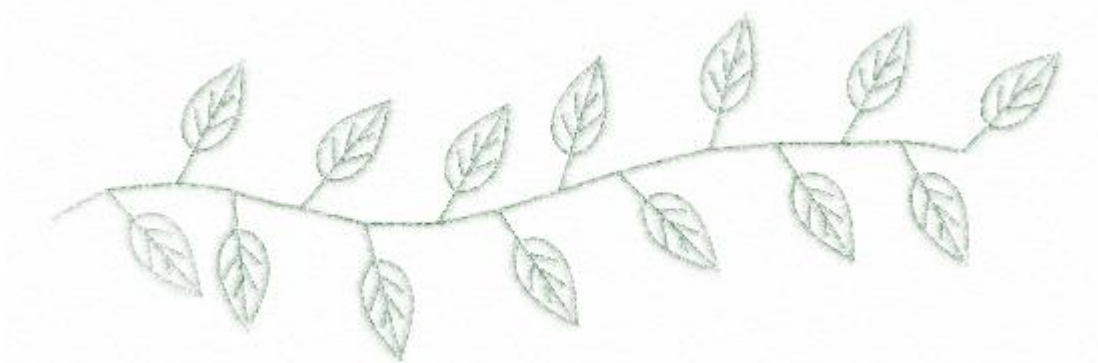
4) Going through the code:

- SetDelay(0) - sets playback to be at maximum speed – we want the leaves to be drawn as fast as possible.
- WaitSampleMouse() - pauses script playback, waiting for the user to click in the canvas. ArtRage will bring up a notification that it is waiting for a mouse-click in the canvas. It also samples the mouse coordinates for use with MouseX() and MouseY() later.
- real rLastX = MouseX() and rLastY = MouseY(); Gets the current mouse location. We're going to be comparing where the mouse is now to where the mouse was as we draw so we're initializing the last known location of the mouse.
- real rStrokeEndX = rLastX and rStrokeEndY = rLastY; Each new segment of the vine starts from the location of the end of the previous segment of vine so the vine has no gaps. Here we're initializing where the last vine segment ended.

- `real rSegLen = 30;` This is how far we want the vine to grow before there's a leaf. Make this number smaller and you'll get leaves sooner on the vine. Make this number larger and there will be more vine before a leaf is drawn.
- `real pi = 3.14159265` – stores the value for pi in a variable called pi. Useful for doing trigonometry.
- `flag fTopLeaf = true;` We draw leaves on alternate sides of the vine. If the last was on the top, the next will be on the bottom.
- `real rDirectionX = 0` and `rDirectionY = 0;` We will need to know the direction of travel of the mouse so we can draw the leaf sticking outwards. For aesthetics we also add in a bit of direction smoothing.
- `real rDX = 0` and `rDY = 0;` To work out the direction we need to know the difference between the previous mouse point and the current mouse point. So we store the differences in X and Y
- `while (MouseButtonIsDown()) {` This is the start of our outer loop. So long as the mouse is down on the canvas, draw vines and leaves. When the mouse goes up, we will stop.
- `<StrokeEvent>` Tell the script engine we're about to start a paint stroke. Each segment of the vine is a single stroke, with multiple event points.
- `StrokeHeader(rStrokeEndX, rStrokeEndY, rLastX, rLastY, MousePressure(), MouseTilt(), MouseOrientation(), MouseInverted());` This calls the `StrokeHeader` function to set up the brush head direction and smoothing for the paint stroke.
- `real rStrokeLen = 0;` We need to track how far along the vine segment we've travelled to know if we've gone far enough to end the vine segment and draw a leaf. So we start by saying we've travelled 0 pixels along the canvas.
- `while (MouseButtonIsDown() && rStrokeLen < rSegLen) {` This is the start of the inner loop to draw chunks of a single vine segment. We want to keep drawing the vine segment so long as the mouse stays down, and the vine length is shorter than the length we said a segment should be.
- `WaitSampleMouse();` Wait for the mouse to move again.
- `real rNewX = MouseX()` and `rNewY = MouseY();` Grab the new mouse position.
- `rDX = rNewX - rLastX` and `rDY = rNewY - rLastY;` Calculate the difference between the previous mouse point and this mouse point.
- `real rVLen = VectorLength(rDX, rDY);` Calculate the distance in pixels between the previous and current mouse points. This is how far in pixels the mouse travelled since the last chunk of vine segment.
- `Loc: (rNewX, rNewY) Pr: MousePressure() Ti: MouseTilt() Ro: MouseOrientation() Rv: NO Iv: MouseInverted();` Add a mouse event to the stroke that makes up the vine segment.
- `rLastX = rNewX; rLastY = rNewY;` This mouse event now becomes the old mouse event so it can be compared to the next mouse event next time we loop
- `rStrokeLen += rVLen;` Add the distance we travelled for this chunk of vine to the total distance we've travelled along the vine.
- `}` This is the end of our inner loop which draws a segment of vine. If the total distance we've travelled for this vine segment (`rStrokeLen`) is still less than the length of a complete vine segment (`rSegLen`), and if the mouse is still down, we loop back to the start of the inner loop.

- `rStrokeEndX = mouseX()` and `rStrokeEndY = mouseY()`; This is the mouse location where our vine segment ends. We keep that so when we start the next vine segment it will start from exactly the same place the previous vine segment ended. This is also the anchor point for the stem of the leaf we'll draw shortly.
- Loc: (`rStrokeEndX`, `rStrokeEndY`) Pr: `mousePressure()` Ti: `mouseTilt()` Ro: `mouseOrientation()` Rv: NO Iv: `mouseInverted()`; - To finish off the end of the vine segment we need to add two more mouse locations. The first time is as though we are drawing with the mouse to the last point, and the second time is the event that would have been recorded when the mouse button was released. Having the two mouse point events makes the sure the stroke finishes on the correct point.
- `</StrokeEvent>` We have completed the stroke which makes up the segment of the vine. Every `<StrokeEvent>` to start a stroke **must** have a corresponding `</StrokeEvent>` to end a stroke.
- Now we can draw the leaf at the end of our vine segment.
- `real rLeafScale = 0.5`; Depending on how large you want your leaves on the vine, and how large the original leaf was when you recorded your script, you might need to adjust this to get an aesthetically pleasing vine. Because we drew the original leaf around 140 pixels in height, and we've got a vine segment length of 30, if we scale the leaf size by half then the distance between two leaves on the same side of the vine is close to the size of one leaf. But mess with this number to get a value that makes you happy.
- `real rLeafRotation = 0.5`; For aesthetics I want the leaves to point slightly in the direction of the growth of the vine. 0 would have the leaves pointing directly out at a right-angle to travel. $\pi / 2$ (approx. 1.57) would have the leaves point in exactly the same direction as the vine and be drawn overtop each vine segment. 0.5 gives a tilt of about 30 degrees toward the growth direction of the vine.
- `if (fTopLeaf) rLeafRotation = pi - rLeafRotation`; Every second leaf starts from the opposite side. A circle is $\pi \times 2$ radians, so half the circle is π , so if we subtract the rotation of the leaf from π it will be on the opposite side of the vine, but still pointing slightly in the direction of travel.
- `fTopLeaf = !fTopLeaf`; If topleaf was true it's now false. Or if it was false, it's now true. This means the next leaf will be drawn on the other side of the vine.
- `rDirectionX = rDirectionX * 0.1 + rDX`; and `rDirectionY`; The direction of travel we say is a bit of the old direction, plus the new direction. This puts in a smidgen of smoothing so the leaf direction will not do a wild direction change on a small travel distance.
- `rLeafRotation -= atan2(rDirectionX, rDirectionY) + 3 * pi / 2`; The actual rotation of the leaf is based on the arctangent of the direction of travel.
- `Leaf(rStrokeEndX, rStrokeEndY, rLeafRotation, rLeafScale, mousePressure())`; The moment we've all been working toward! Draw one leaf. The anchor point for the stem is at `rStrokeEndX`, `rStrokeEndY`. Our Leaf function uses the anchor point for the scale and rotation of the leaf, and starts drawing the stem from this point. `rLeafRotation` and `rLeafScale` give the orientation and size of leaf, and these values change the drawing space of the Leaf function so the scripted leaf is drawn with a new size and rotation. And finally we pass in the current mouse pressure so the leaf can be drawn heavier or lighter depending on the stylus pressure of our graphics tablet.

- 5) Now if you save your MyVine.arscript and run it in from the ArtRage actions panel, you should get a vine drawn in the current colour with the current tool. For testing, first make sure you have the pencil tool selected, and that you can make a stroke on the canvas before you run the script. At least then you'll know you don't have the tool settings wrong or you're drawing with the colour white on a white canvas (you'd be surprised how often I make that mistake!). All going well you should get something that looks like the following:



- 6) If it's all gone horribly wrong... The ArtRage script engine should bring up a message box for most script errors telling you approximately where it thinks the error is. The most likely problem is mistyping a variable name. Another common problem is missing the '}' to close off the loop blocks. If you absolutely can't get your script to run check that the Vines.arscript included with this tutorial runs (You can drag and drop Vines.arscript onto the ArtRage canvas to run it as a script). If it does run correctly you can compare your script code to the Vines.arscript script code. Cut'n'paste your way to programming success!
- 7) And if it goes right then we can play around with our script and make it even better!
- How about using the stylus pressure to change the scale of the leaves?
`SetTransScale(0.1 + rScale * rPressure , 0.1 + rScale * rPressure);` High pressure makes bigger leaves, light pressure makes smaller ones.
 - What about recording a few different shaped leaves, and maybe a flower, and randomly using a different shaped leaf or flower?
 - `int n = Random(5);`
 - `if (n == 0) { LeafA(...) }`
 - `else if (n == 1) { LeafB(...) }`
 - `else if (n == 3) { LeafC(...) }`
 - `else { Flower(...) }`
 - What about adding a `while(true){` loop outside the existing outer loop, so the vine script keeps running until someone hits the escape key? Put the `'while(true) {'` just after the `SetDelay(0)` line so new vines start from the mouse-down point, and put the closing `'}'` after the previous `'}'` at the very end of the script.

Conclusion

You can use this tutorial as a basis for writing any ArtRage script which draws along with the mouse or graphics tablet stylus. You can add events along a stroke, change the parameters that the ArtRage stroke engine gets. What about a stroke that draws along in loops and spirals as you draw? The possibilities are endless. Remember that you can change tool settings, colours, tools, and anything which can be recorded in a script. If there's something you want to mess with, record a script or an action with an example of what you want to change to see what the script engine produces, then change some of the recorded values to variables you can change as you go.

