Computer generated 3D strokes

Industrial Engineering Final Project

Enric Forés Solar
Tutor: Ning Xie
Shanghai, 2015

同濟大學
TONGJI UNIVERSITY
UPC
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA

INDEX

FOREWORD

The motivation for carrying out this project in the field of computer generated graphics comes from trying to merge previous studies on il·lustration with the present studies of industrial engineering.

On one hand my passion for drawing has given me an idea of which tools, processes and aestethical criteria are used by artists to create images. On the other hand my technology training has provided me new points of view to solve problems and understand systems and, specially usefull for this project, some knowledge of python language.

The 3D computer graphics software used is Blender. It has been chosen because it's free and open-source, it covers all the needs of a complete animation pipeline and, most of all, for the large online community and documentation. Because of this documentation format most of the bibliography is only hung at internet without printed editions available.

Initially the idea was to explore the possibilities of Blender to generate watercolor and ink effects using its non-photorealistic renderer "Freestyle". Thanks to the help of Lluis Solano Albajes, from Computer Science department in ETSEIB-UPC, and Antonio Susin Sanchez, from Applied Mathematics department in ETSEIB-UPC, an investigation proposal around this objective was made. The opportunity offered by Tongji University to do the project there was great to get in touch with the tutor of this project, Ning Xie, to whom I have to thank for guiding me into the front-end field he explored during his PHD and letting me explore the possibilities of his amazing software.

INTRODUCTION

Stroke-based rendering (SBR) is an approach to creating non-photorealistic images (NPI) by placing discrete elements called strokes, such as paint strokes or stipples.

Several lines of investigation are being followed to obtain software that authomatises this process, both in still and moving images

The aim of this present work is to adapt the Computer Generated Images (CGI) part of a common 3D animation pipeline (concatenated processes with which a 3D animation is made, from the concept art and sketches to the final compositing of the scenes) to apply the authomatically generated stroke textures of PHD Ning Xie's software on the 3D models.

Ning Xie's software generates, given an image and the contour of the desired stroke, a computer generated stroke extracting the colour information from the given image. Originally programmed to emulate sumi-e oriental drawings (traditional style that represent the motive with few strokes) in this work a way will be presented to bring this technology from 2D to 3D.

For the purpose of this thesis a simple animation of a bamboo leaf has been created to work on a simple example while developing the process .

The main (and original) challenge of this work is writing a python script to extract from blender the desired contours of the strokes in a way that can be used by Ning Xie's software.

It's important to comment that NPR is nothing near to a science with clear objectives as the results of it only can be evaluated using aestethical criteria. However the tools and technology used to achieve it has a clear and exact definition in which this thesis will work on.

STATE OF ART

STROKE BASED RENDERING

An investigation has been done on the actual technologies and algorithms used to create automathically generated NPI, specially existing solutions for SBR, an automatic approach to creating nonphotorealistic imagery by placing discrete elements such as paint strokes or stipples.

Researchers have proposed many SBR algorithms and styles such as painting, pen-and-ink drawing, tile mosaics, stippling, streamline visualization, and tensor field visualization.

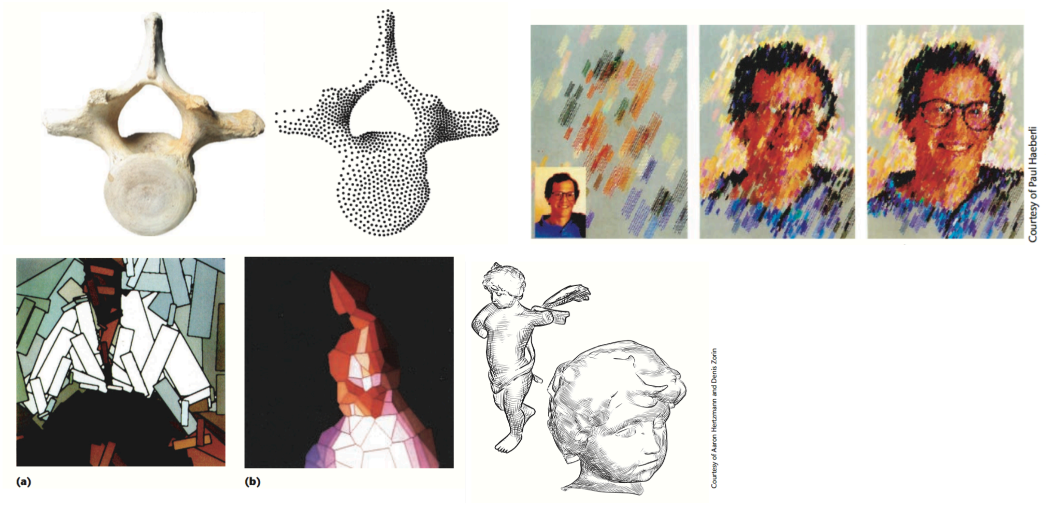

Courtesy of Paul Haeberli
(a)
(b)
Courtesy of Aaron Hertzmann and Denis Zorin

Figure 1. Several examples of Stroke Based Rendering

An institution that nowadays is pushing hard the fronteir of knowledge in this field is Disney Research Zurich. In “Authoring and animating painterly characters” they show a system in which 3D stroke-based paintings can be deformed using standard rigging tools. They also propose a configuration-space keyframing algorithm for authoring stroke effects that depend on scene variables such as character pose or light position. Their primary technical contribution is a novel interpolation scheme for configuration-space keyframing that ensures smooth, controllable results.

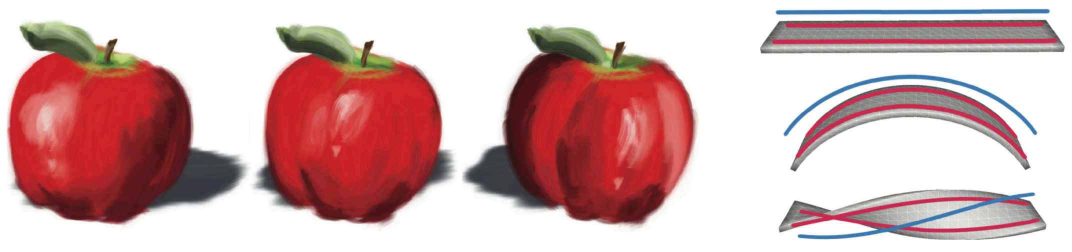


Figure 2. Figures from “Authoring and animating painterly characters”.

This is a good example where automathically generated strokes could speed up the process. In their workflow an artist has to digitally paint one by one each stroke that later will be deformated and shown upon the 3D object. A contribution that Ning Xie’s stroke generator software could do would be to do this artists task. This way parametric strokes could be programmed, for example, to texture hundreds of leaves of a tree with a different stroke for each without needing a an artist to texture each of them.

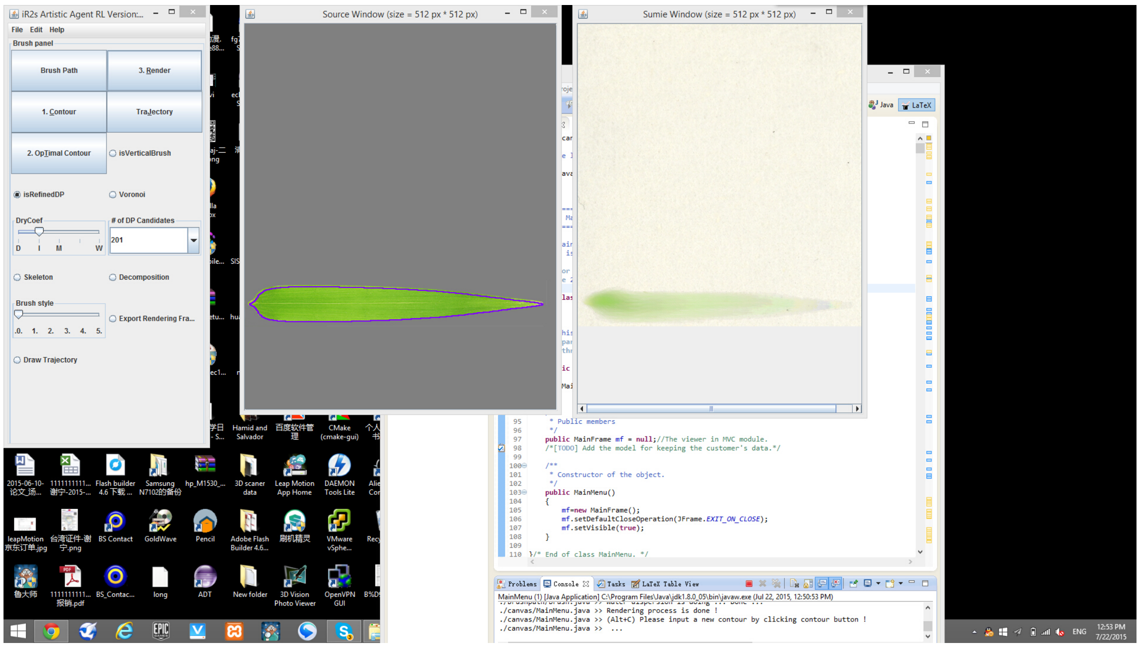
2D STROKE GENERATOR

In 2012 Ning Xie publishes his thesis “A machine learning approach for automatic stroke generation in oriental ink painting” from which two paragraphs will be quoted:

“Stroke placement is a common problem in painterly rendering. The study of stroke placement is how to generate stroke with realistic brush texture in a desired shape. It is widely used to digital painting artwork generation. The state of the art methods can efficiently map a scanned brush texture by deforming (cutting and pasting) them onto a user-drawn path or the destination shape. Users cannot get strokes with shapes similar to their expectation. Moreover, the processing of distorting the shape from stored one to desired one and non-natural stretching texture causes unsatisfactory defects such as undesirable folder or creased appearance inside the corner or curve.

In this dissertation, we propose a highly practical framework for generating the expressive appearance of brush strokes. We introduce the intelligent learning agent theory into the painterly rendering problem. A brush is modeled as an intelligent agent under Markov decision process (MDP). We elaborate on the design of actions, states, and rewards tailored for the brush agent. We motivate stroke placement problem as learning to optimally select actions by a robot (agent) to cover given regions of desired strokes. We propose two optimal policy learning approaches the model-based method and the model-free method, to return control policies in the reinforcement learning framework. We also propose methods to understand a scene in a 2D real photo and segment it into regions that can be covered by single strokes based on computer vision. “

Ning Xie’s thesis cristalises in a java application with which the user can generate multiple kinds of strokes using several parameters as dampness and style.



```
     * Public members
     */
    public MainFrame mf = null;//The viewer in MVC module.
    /*[TODO] Add the model for keeping the customer's data.*/

    /**
     * Constructor of the object.
     */
    public MainMenu()
    {
        mf=new MainFrame();
        mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mf.setVisible(true);
    }

}/* End of class MainMenu. */
```

```
MainMenu (1) [Java Application] C:\Program Files\Java\jdk1.8.0_05\bin\javaw.exe (Jul 22, 2015, 12:50:53 PM)
./canvas/MainMenu.java >> Rendering process is done !
./canvas/MainMenu.java >> (Alt+C) Please input a new contour by clicking contour button !
./canvas/MainMenu.java >>  ...
```

Figure 3. Interface of Ning Xie's Java application.

The input that Ning Xie's software needs to generate the strokes are the original image and the contour of the stroke within that image that the user wants to create.

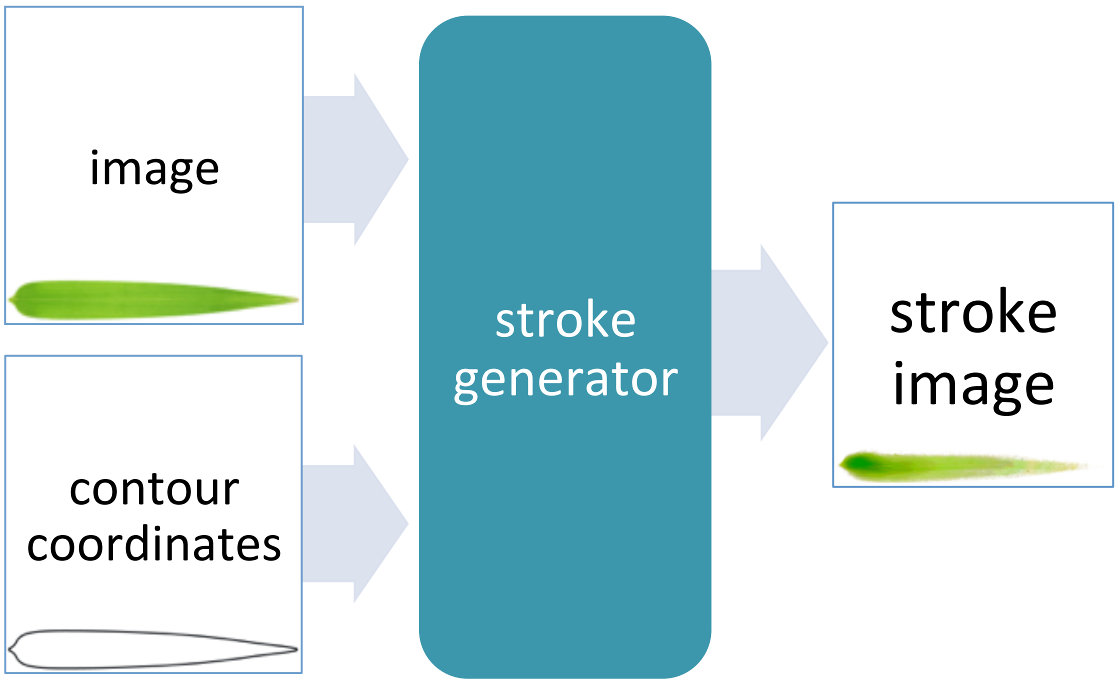


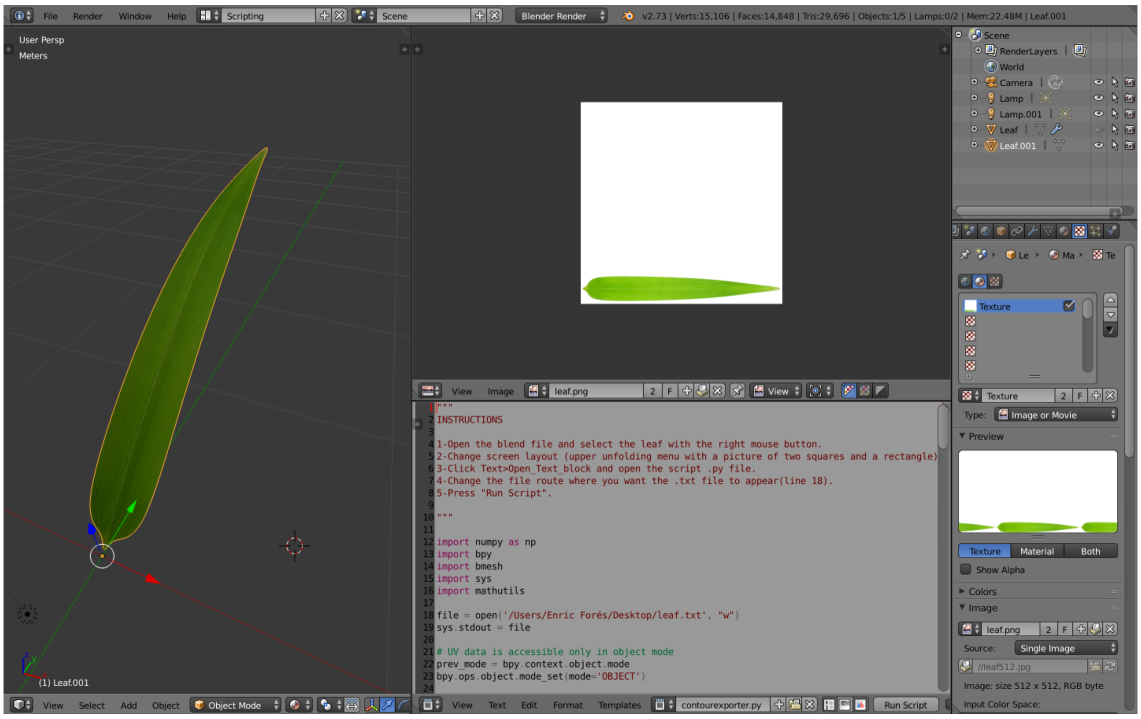
Figure 4. Block diagram of Ning Xie's Java application inputs and outputs.

3D SOFTWARE SCRIPT

To achieve our goal of bringing Ning Xie's software capabilities to 3D texturing open source Blender 2.73a 3D software has been used. Blender has a python editor that allows to execute scripts like the one needed.

In the case of study (three dimensional scenery) this "original image" that has been used is the parametrical or custom painted texture of the model. To apply a texture to a model the grid of vertexs that build its mesh are flattened on a 2D plane, usually a 2^nx2^n square bitmap to improve rendering time in later processes. This plane's coordinates are called u and v to differenciate them from the cartesian coordinates x, y and z from the 3D environment where it will be applied.

In Ning Xie's software the contour input is generated by the user using a mouse or a digital tablet. In the present case the objective is to generate the contour with the same tool that manages the texture: the 3D computer graphics software. Blender includes a python console and a script editor. To generate a text file with the uv coordinates as the input of Ning Xie's software an extractor script has been created.



```
 1 """
 2 INSTRUCTIONS
 3 
 4 1-Open the blend file and select the leaf with the right mouse button.
 5 2-Change screen layout (upper unfolding menu with a picture of two squares and a rectangle)
 6 3-Click Text>Open_Text_block and open the script .py file.
 7 4-Change the file route where you want the .txt file to appear(line 18).
 8 5-Press "Run Script".
 9 
10 """
11 
12 import numpy as np
13 import bpy
14 import bmesh
15 import sys
16 import mathutils
17 
18 file = open('/Users/Enric Forés/Desktop/leaf.txt', "w")
19 sys.stdout = file
20 
21 # UV data is accessible only in object mode
22 prev_mode = bpy.context.object.mode
23 bpy.ops.object.mode_set(mode='OBJECT')
24 
```

Figure 3. Interface of 3D computer graphics software Blender in scripting mode.

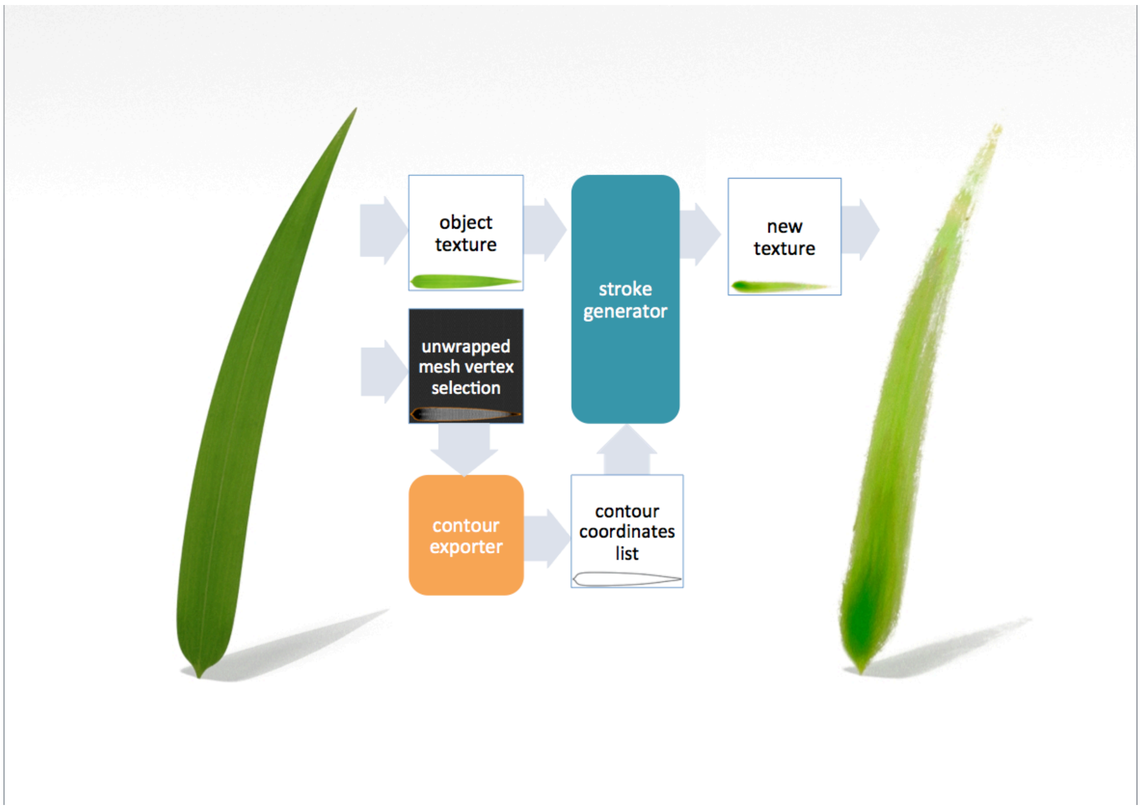


Figure 4. Block diagram of Ning Xie's Java application and the contour exporter inputs and outputs.

The contour exporter python script works on a textured 3D object. This object consists on a mesh and it's texture. This texture is allready unwrapped upon the original texture where the user needs to select the loops (each vertex of the mesh can generate several loops in the uv plane once unwrapped) that will conform the contour of the generated stroke.

Here is presented the script with extended comments in green:

```
""""
INSTRUCTIONS

1-Open the blend file and select the leaf with the right mouse
button.
2-Change screen layout (upper unfolding menu with a picture of
two squares and a rectangle) to Scripting.
3-Click Text>Open_Text_block and open the script .py file.
4-Change the file route where you want the .txt file to
appear(line 18).
5-Press "Run Script".

""""

# bpy and sys libraries are needed to access the system and
# blender environment. The other three libraries are used for
# their buit-in classes and mathematicall operations related.

import numpy as np
import bpy
import bmesh
import sys
import mathutils

# a .txt file is created and the output redirected to it.

file = open('/Users/Enric Forés/Desktop/leaf.txt', "w")
sys.stdout = file

# UV data is accessible only in object mode

prev_mode = bpy.context.object.mode
bpy.ops.object.mode_set(mode='OBJECT')

# Update vertex selection properties, in case the script wasn't
# run in object mode

bpy.context.object.update_from_editmode()

# Active object assumed to be a mesh and already have a UV map

mesh = bpy.context.object.data
```

```
uv_layer = mesh.uv_layers.active.data
selected_loops = []
selected_vertices = set()

# First a list of uv coordinates of the selected vetices is
# arranged:

for index, uv_loop in enumerate(uv_layer):
    if(uv_loop.select):
        selected_loops.append(index)

for loop_index in selected_loops:
    selected_vertices.add(mesh.loops[loop_index].vertex_index)

uv_coord = []

for poly in mesh.polygons:
    for loop_index in range(poly.loop_start, poly.loop_start +
poly.loop_total):
        if mesh.loops[loop_index].vertex_index in
selected_vertices:
            vec =
mathutils.Vector((round(uv_layer[loop_index].uv[0]*512),
round(uv_layer[loop_index].uv[1]*512)))
            if vec not in uv_coord:
                uv_coord.append(vec)

# At this point the task that is left is to order the pixels by
# proximity making the list to go over the contour while adding
# the connecting pixels between each uv coordinate and the next.

# This way of ordering the points restricts the shape of the
# contour; a stroke with a very slim central region can give
# problems as the track being generated may jump to the other
# side of the stroke at that point instead of following the
# contour.

# Function that returns the distance between two pixels:

def distance(pt_1, pt_2):
    pt_1 = np.array((pt_1[0], pt_1[1]))
    pt_2 = np.array((pt_2[0], pt_2[1]))
    return np.linalg.norm(pt_1-pt_2)

# Function that returns, given one point, the closest one from a
# list:

def closest_node(node, nodes):
    pt = []
    dist = 9999999
    for n in nodes:
        if distance(node, n) <= dist:
            dist = distance(node, n)
            pt = n
    return pt

# Next function is the Bresenham’s algoritm, which given two
# points it returns the pixels of the straight line that
# connects them

def line(x0, y0, x1, y1):
        points_in_line = []
        dx = abs(x1 - x0)
        dy = abs(y1 - y0)
        x, y = x0, y0
        sx = -1 if x0 > x1 else 1
        sy = -1 if y0 > y1 else 1
        if dx > dy:
            err = dx / 2.0
            while x != x1:
                points_in_line.append((x, y))
                err -= dy
                if err < 0:
                    y += sy
                    err += dx
                x += sx
        else:
            err = dy / 2.0
            while y != y1:
                points_in_line.append((x, y))
                err -= dx
                if err < 0:
                    x += sx
                    err += dy
                y += sy

        return points_in_line

#With these three last functions the contour is generated:

path = []
numuvs = len(uv_coord)
path.append(uv_coord[0])
uv_coord.remove(uv_coord[0])
i = 0
while i != (numuvs-1):
    previous = path[i]
    closer = closest_node(previous, uv_coord)
    path.append(closer)
    uv_coord.remove(closer)
    i = i + 1

i = 0
while i != (numuvs-1):
    previous = path[i]
    next = path[i+1]
    filled = line(previous[0], previous[1], next[0], next[1])
    for mathutils.Vector in filled:
        print(int(mathutils.Vector[0]),
int(mathutils.Vector[1]))
    i=i+1

previous = path[-1]
next = path[0]
filled = line(previous[0], previous[1], next[0], next[1])

for mathutils.Vector in filled:
    print(int(mathutils.Vector[0]), int(mathutils.Vector[1]))

# Restore whatever mode the object is in previously
bpy.ops.object.mode_set(mode=prev_mode)

sys.stdout = sys.__stdout__ #reset

file.close()
```

AREA OF APPLICATION

The area of application of this script is in CGI pipelines where a SBR style is required without making an artist draw the stokes. Here there are two hypothetical cases:

- A 3D animation during a part of iwhich a painterly style is necessary; the objects can be used only changing the textures.

- Any case whan a large amount of strokes must be made, for example, when creating the leaves of a tree. In this case the software can be programmed to generate a great number of different leaves.

FUTURE WORK

The future work that could be done towards improving Ning Xie's code capabilities in a 3D environment is , in my opinion, translating it from Java to Python and implement it as an add-on. This would simplify a lot the workflows as it could be fully automathic and insertable in other sripts facilitate the participation of CGI developers on finding new aplications such as:

- Design a way, maybe a mask, to simulate the building of the stroke, begining when the brush touches the paper till it finishes the stroke.

- Find a way to sincronise the painting of the stroke with a 3D object (brush) that leaves a path.

- Use particle systems to enhance the effect of the ink spreading in the air.

CONCLUSIONS

The objective of the project has been achieved but the script is far from being free of bugs (for strange contour shapes) and some more programming would be nice to improve usability.

It’s remarcable how online Blender and python developers communities have helped to carry this project out. It couldn’t be made without them. A good conclusion would be to upload the script to one of these forums to thank all the free resources.

Personally, I have much enjoyed getting to taste what would the job of a studio technician be tinkering around with the pipeline. It has been an honor to work with such a new field as Ning Xie’s software and I hope to be able to do it in the future.

BIBLIOGRAPHY

REFERENCE ARTICLES

-A MACHINE LEARNING APPROACH FOR AUTOMATIC STROKE GENERATION IN ORINENTAL INK PAINTING
A Thesis in Department of Computer Science Graduate School of Information Science and Engineering Tokyo Institute of Technology by Ning Xie

-Bassett, K., Baran, I., Schmid, J., Gross, M., and Sumner, R. W. 2013. Authoring and animating painterly characters. ACM Trans. Graph. 32, 5, Article 156 (September 2013), 12 pages. DOI: http://dx.doi.org/10.1145/2484238

-A Survey of Stroke-Based Rendering July/August 2003
Aaron Hertzmann
Dept. of Computer Science University of Toronto

-Painterly Rendering for Animation
Barbara J. Meier
Walt Disney Feature Animation

ONLINE REFERENCES

Blender Manual: http://www.blender.org/manual/

http://stackoverflow.com/questions/7789154/can-blender-export-per-vertex-uv-coordinates

http://www.blender.org/api/blender_python_api_2_63_release

http://blenderartists.org/forum/showthread.php?220171

http://wiki.blender.org/index.php/User:Pkrime/TexturedStrokes

https://cgcookie.com/blender/cgc-courses/introduction-python-scripting-blender

ANNEXES

The CD includes the following materials:

- Python script in .py format.
- The present report in digital format.
- Blender .blend file used for the examples.
- Short clip showing a 3D animation with the result of the example.