EXPLORING OPTIMIZATION OF STROKE-BASED RENDERING OF 3D SHAPES *

Darya Todoskova
Bachelor Student
École Polytechnique
darya.todoskova@polytechnique.edu

Lucas Serog
Bachelor Student
École Polytechnique
lucas.serog@polytechnique.edu

under the supervision of

Prof. Marie-Paule Cani, Ph.D.
Project Supervisor
LIX, CNRS/École Polytechnique
marie-paule.cani@polytechnique.edu

Renaud Chabrier, Ph.D.
Project Supervisor
LIX, CNRS/École Polytechnique
renaud.chabrier@m4x.org

Tara Butler
Ph.D. student
LIX, CNRS/École Polytechnique
butler@lix.polytechnique.fr

ABSTRACT

Handmade drawings and 3D shapes often belong to two separate worlds. A drawing consists of strokes on paper that are simply visible by one's eyes, and a 3D object can only be seen using virtual cameras, shaders, and projective geometry. But the link that joins both these worlds arises when attempting a sketch-based modeling approach (i.e. an intuitive and effective method for creating a 3D model from 2D sketches). The purpose of this project is to explore the creation of a 3D model without any prerequisites. All we will need from the user are curvature indications on a sketch representing a smooth, unknown shape.

***K*eywords** Curvature · Polylines · 2D Strokes · Projection · Cameras · Skeleton · Tubular · WebGL · Three.js

1 Introduction

In this project we explore bending lines of 2D sketches. These lines are meant to convey surface bending directions, to show curvature variations. Firstly we study their classification, then we go in details on the way of inferring the desired 3D surface from the strokes by making the chosen segment "protrude" or "intrude", and finally we compare the result to the expected shape.

Our main contribution is thus the introduction of a complement to the existing sketch-based modeling method used in Matisse (described in 2.2) : a sketch tool that enables the user to enhance the 2D sketch with additional curves (or bending lines) in order to reveal the major bending directions of the free-form shape. This additional feature thus allows the user to draw different curvatures in a single-view mode, since Matisse only supported drawing the final object by combining multi-view inputs (by rotating then adding the different portions of the model).

*CSE303 Computer Science Project Report

1.1 Motivation

The initial motivation for this project was to allow people with little or no artistic training to be able to easily visualize their ideas in 3D.

With our work, we aim to contribute to stroke-based 3D rendering by going about three different aspects of the field. Firstly, we wanted to find a meaningful way to identify and cluster drawn curves of a given shape. This study would then be used in our algorithm to process each region according to its strokes and modify its depth values into the corresponding 3D shape. Finally we want to test the user's perception of the meaning of bending lines by comparing the originally generated 3D shape to the one inferred by the user's strokes.

These are features that would come to good use by enabling users to get 3D models of their imagined shape by inputting extra information to the system at any unknown scale. Our framework thus provides a useful complement to existing sketch-based modeling methods.

1.2 Major Challenges

The first challenge we encountered was the lack of a distinctive convention of sketched curves to indicate shape variations in a 2D drawing. We thus had more flexibility in establishing our own rules, while considering both the interface's existing algorithmic structure, and the human perspective. Therefore, before the implementation in WebGL, we looked at scientific drawing techniques and attempted to uncover any laws or patterns to determine the best way to represent the 3D model in this rendering approach. We also conducted a few perceptual user studies to get direct human feedback.

Another major difficulty arose during the programming of the different algorithms. Since we were using a Three.js Blobtree library (described in 2.1) developed by Dioxygen Software for Dualbox and Laboratory LIX, we had limited documentation. Moreover, managing different aspects of the code in JavaScript was quite challenging due to our lack of fluency in the language, despite the different tutorials [2] [3] we had done in the early stages.

1.3 Goal

Our goal is to allow users to model a simple free-form shape in 3D by sketching sparse 2D strokes onto the non-photo-realistic rendering (described in 2.3) of an arbitrary shape generated by the shape generator (described in 2.1). The user would sketch strokes representing the major bending directions onto the 2D rendering in order to achieve the 3D object originally generated on the interface. Once the strokes are validated by the user, a 3D object would be derived from the sketch with the specific indicated curvature pattern. By making the user try to recreate the originally generated 3D shape through strokes on a 2D contour of the shape, we not only create a system that allows the user to draw different curvatures in a single-view mode, we also validate (or reject) our understanding of the interpretation of strokes on 2D sketches, through user perception, in order to eventually define a clear methodology to distinguish and cluster them.

2 Previous work

2.1 Blobtree library and Shape Generator

Recent research has shown that sketch-based modeling methods are most effective when targeting a specific family of surfaces, which is why we are going to target the shapes generated by the shape generator that was developed. It gives us an infinite number of arbitrary shapes that can then be displayed using the non-photorealistic rendering introduced by Renaud Chabrier 2.3, on which the user will draw the curvature strokes.

We decided, for efficiency and simplicity purposes, that we would only consider tubular shapes, whose skeletons are poly-lines composed of three segments (i.e. skeletons with three bones) back-to-back. Indeed, we noticed that round shapes wouldn't allow us to portray bending variations through strokes, and that three segments was sufficient to display the functioning of our method.

Our algorithm, just like the shape generator, is based on the three-js-blobtree library [4], which implements a Blobtree representation [1]. The latter was used in the shape generator to first create a 3D segment, whose generative function's

[2] https://imagecomputing.net/damien.rohmer/teaching/mpri2-39/TD/threejs/content/000_threejs_tutorial/index.html

[3] https://threejs-journey.com/?c=g1&gclid=Cj0KCQjwmouZBhDSARIsALYcouq_VN61C5t36P4ARvlbujviUwnI9uzFp7iiF4nI4WzXRa_xUq36KRQaAkQYEALw_wcB#

[4] https://github.com/dualbox/three-js-blobtree

arguments are the two vertices of its extremities, the density (constant to modulate the implicit field), the primitive material (color) and the volume type. There are two options for the volume type : a *weighted distance* function along the skeleton or the *SCALIS* convolution function [2]. However, using a weighted distance function in the convolution function has drawbacks: the influence of the weight diminishes as the degree of the kernel increases. That is why the shape generator uses the *SCALIS* convolution function.

2.2 Matisse

Matisse [3] is a modeling system that has been implemented in Javascript in 2019. With the help of this software a user can easily and interactively paint 2D regions of any topology to model free-form shapes. While painting, it is possible to change the zoom factor as well as the view-point. For the conversion into 3D shapes, implicit modeling that fits convolution surfaces to regions is used.

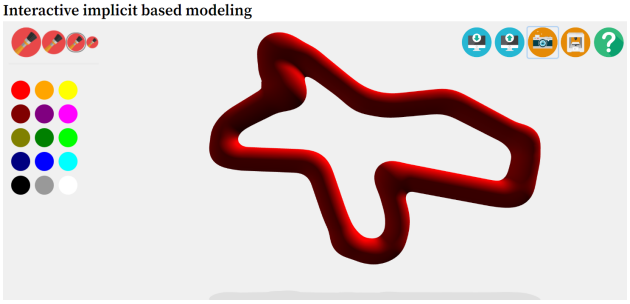

Figure 1: 3D shape created with Matisse software

2.3 Non-photo-realistic Rendering

The non-photo-realistic rendering introduced by Renaud Chabrier [4] [5] is a difference between two images. The rendering is done by taking two different images (obtained by translation) of the object, superposing them, and then taking the difference of the two. In order to be as close as possible to a real life scenario, the translation is done such that it represents the distance between the human eyes. The difference between the two images is then computed pixel by pixel, which displays the contour of the image.

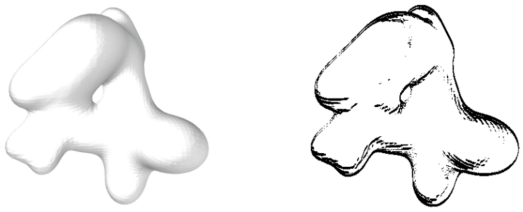

Figure 2: General result of Non-photo-realistic Rendering

3 Perceptual User Study

3.1 Preparation

Classifying sketched strokes on a 2D contour requires overall studying the relationship between 2D sketches and 3D shapes.

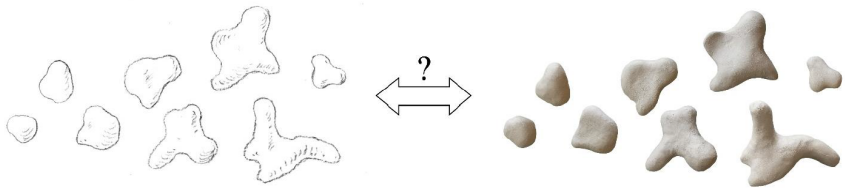

Figure 3: Relationship between handmade drawings and 3D shapes

This is why we decided to conduct a user study with handmade sketches. Our goal was to uncover any rules or patterns that would allow us to determine the best way to utilise the strokes. Due to our lack of artistic knowledge and online documentation, we had to perform a couple user studies before achieving one that gave us meaningful results.

3.1.1 First User Study : Conclusion

Results of our first perceptual user study are available in Appendix A. After carefully analysing the results, we noticed that round shapes weren't going to allow us to portray intrusion or protrusion. Indeed, a circular contour with strokes on the surface will always be seen as a sphere. We thus decided to work with tubular shapes, and draw bending strokes on the corners between two segments.

3.2 Second User Study: Depth Perception

Hypothesis: People interpret the interior of a stroke as an indication of the side that protrudes/intrudes.

In order to test this, we created another questionnaire with ten drawings, this time with five unique tubular shapes.

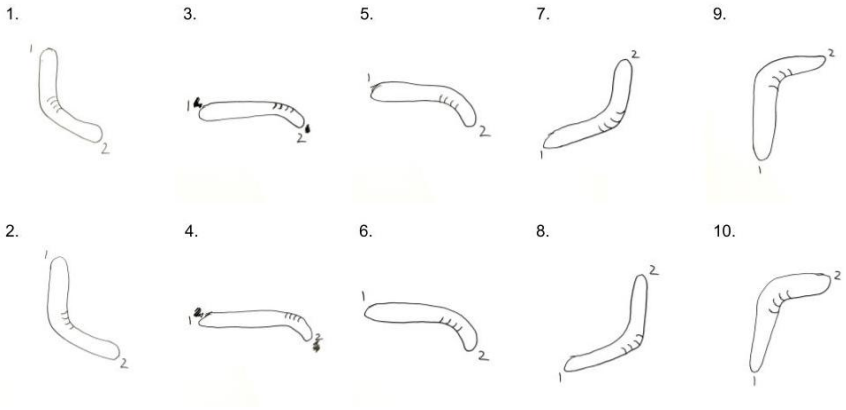

Figure 4: Sketches used in second user study

For each shape, there are two drawings in which there are four strokes placed in the exact same spot, with the difference being the strokes facing opposite directions. By changing the direction of the strokes, we were hoping to change the segment that protrudes or intrudes. Every drawing has two ends labeled 1 and 2, and the goal for subjects was to determine which end seemed to be protruding towards them. Once again, subjects were only allowed to choose between two different answers, end 1 or end 2.

3.2.1 Results

Some quick numbers on the subjects in our user study:

- Participants: 66
- Gender: 42% male, 58% female
- Age: 17% ($\leq$18), 83% (19-29)

Below is a table with the results of our second questionnaire. The models with an asterisk have strokes whose interior is facing the end of the tube marked "1". The ones without have strokes whose interior is facing end 2. (Figure 9 in the appendix explains the notion of the interior of a stroke.)

Model Number	Percent that Perceived End 1 in Front	Percent that Perceived End 2 in Front
1.	13.6%	86.4%
*2.	37.9%	62.1%
*3.	77.3%	22.7%
4.	21.2%	78.8%
5.	7.6%	92.4%
*6.	71.2%	28.8%
7.	43.9%	56.1%
*8.	89.4%	10.6%
*9.	95.5%	4.5%
10.	12.1%	87.9%

Figure 5: Results of our second user study

Besides the first shape (more specifically model 2), most subjects perceived the end facing the inside of the curve to be protruding towards them.

3.2.2 Conclusion

With this second user study, we aimed to understand if strokes can portray depth intuitively and how the direction of curvature can be deduced from the curve's interior and the extremity of a segment. Firstly, we noticed that the placement of the strokes, the inner (concave) or outer (convex) part of the bent model, does not affect people's perception of depth in our sketches. This is backed when comparing the shape of model 3 and 4 versus model 5 and 6, as well as the last two shapes (models 7 and 8 versus 9 and 10). Now, in our results, we have one case that is not coherent with the rest. That is model 2. Although the interior of the strokes are facing end 1, only 37.9% of subjects viewed end one to be protruding. However, the rest of the sketches' results support the hypothesis quite convincingly and allow us to conclude with the following two statements:

- If the curvature stroke's interior is pointing towards the extremity of a segment, then that segment is protruding.
- If the curvature stroke's interior is pointing away from the extremity of a segment, then that segment is intruding.

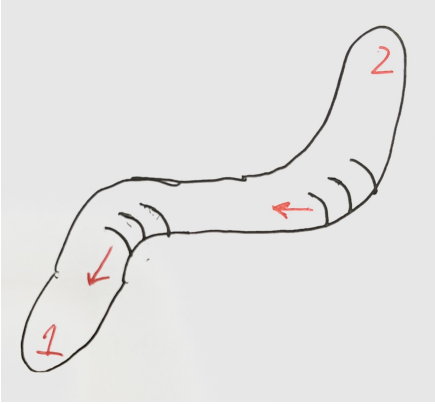

Figure 6: *End 1 is protruding:* the stroke's interior is facing it. *End 2 is intruding:* the stroke's interior is pointing away.

4 Algorithm Overview

In this section, we are going to describe the different steps of our algorithm. Now that we have established rules concerning the interpretation of curvature strokes (as seen in 3.2.2), we have to implement it in WebGL, a JavaScript API for rendering interactive 2D and 3D graphics.

We thus separate our algorithm into the following 7 steps, which we will further describe in detail:

1. Scale camera and scene
2. Display the skeleton
3. Create "pen tool" to draw strokes
4. Associate set of strokes to a segment
5. Determine direction of curvature
6. Perform "flat" 3D rendering of 2D sketch
7. Perform bending on "flat" 3D shape

4.1 Scaling

Although scaling is mentioned as the first step, it was done progressively throughout the project, as you will see below. It was a very important topic to agree on, and required the most research.

4.2 Skeleton Display

The skeleton is a 3D object, seen through a perspective camera, since its points are two-dimensional. Perspective cameras follow a projection mode designed to mimic the way the human eye sees.

The shape generator 2.1, based on the Blobtree architecture, creates segments by connecting two vertices with positions in a Cartesian coordinate system. To trace the skeleton, we retrieved the list of segments (in our case, only three segments) and created *THREE.Line* instances equipped with a *geometry* and a colored *material*. Each line, of different random color, is then added to the scene.

The skeleton of a generated shape not only allows to debug visually on the surface, but also to have defined bones that can be linked to the curved strokes before performing distance computations.

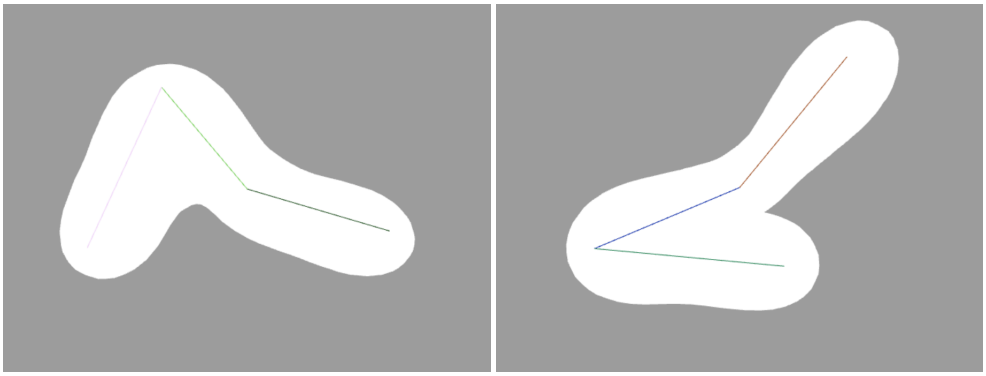

Figure 7: Two Generated 3D Shapes with Visible Skeletons

4.3 Drawing Strokes

Note that we assume that all sketched strokes are C-shaped and are two-dimensional.

The existing script of the non-photorealistic renderer (NPR) 2 uses two perspective cameras on its canvases to display and orbit around the shape. However, in order to draw strokes on the 2D rendering, we need to represent three-dimensional objects in two dimensions. For that, we decided to attach a camera that uses orthogonal projection. With this orthographic camera, all the projection lines are orthogonal to the projection plane. Hence, with the right settings, we are able to draw the strokes directly on the canvas.

In order to draw the strokes indicating bending of the shape, we register all the points through which the mouse cursor goes through on the image when it is clicked and dragged. For scaling purposes, the user draws in the $\{[-1,1],[-1,1],[-1,1]\}$ space, which is WebGL's default space. Then, when we get the positions of the points of the user strokes, the coordinates are in the $\{[0,width],[0,height]\}$ plane. These position coordinates (with a null z-value) are stored in a list which is then rendered on the scene as *THREE.Line* instances equipped with a *geometry* and *material* resulting in poly-lines, similarly to the skeleton. However, before drawing them (i.e. adding the points to the line), we have to shift them and scale them back into the $\{[-1,1],[-1,1],[-1,1]\}$ space. This thus allows us to transform the coordinates from the image space to the space of the orthographic camera.

For the user to draw a set of identical strokes indicating the same region of curvature, a set of three identical strokes are drawn automatically, for simplicity and visual appeal.

4.4 Determining Targeted Segment(s)

For this step and the next, we need to compute distances between points. For that, we need the position of the vertices of the skeleton's segments in 2D, since they are originally in 3D.

We thus use the *project* function, using the same perspective camera used to draw the skeleton, and obtain vertices with 2D coordinates (the z-values are either null or constant).

In order for both points of the middle segment to have the same z-value, we want the View vector of the camera to be aligned with a normal vector. We thus do the following. Once the shape is drawn, the plane will be defined by a normal vector and one of the two endpoints of the skeleton's segment, and hence we obtain the desired plane.

With these projection and camera settings, our tubular 3-segmented generated shapes thus always have their middle segment in the *(x,y)* plane. This implies that only the two segments on the extremities of the shape can be "curved", i.e. protruding or intruding. In the end, a segment that is protruding would have its outermost vertex with a negative z-value, and vice versa.

Since we have established that the middle segment of the shape remains in the plane (has a null z-value), we first need to determine which of the other two segments is targeted by the set of curvature strokes. The targeted segment is the one whose z-value (representing depth in our orthogonal plane) at the extremity of the shape, will be modified. The said segment would, in consequence, be protruding or intruding.

Our method firstly consists in computing the middle point between the two endpoints of the stroke. Since the shape's skeleton contains segments of different colors, we perform a *readRenderTargetPixels* (the render target being the one in which the skeleton was drawn) starting from the aforementioned middle point until a pixel of color is found. The associated segment would thus be identified by its color, and would be the targeted one.

4.5 Determining Direction of Curvature

Note that this project solely determines in which direction the segment bends (either protruding or intruding), and not the magnitude or "amount" of curvature.

We use the results of the perceptual user study 3.2.2 to determine the direction of curvature (targeted segment protruding or intruding). We thus had to find computational method to refer to "the interior of the curve facing/opposing a segment extremity". We achieved this below.

Once we have identified the segment targeted by the set of strokes, we determine the direction of curvature. In the previous step, we computed the middle point between the two endpoints of the curve. In addition, we compute an approximation of the middle of the same two endpoints, but this time *on* the curve. Let A and B be those two points respectively. We then compute the Euclidean distance between point A and the extremity (vertex that is only linked to one segment) of the targeted segment. We do the same for point B. The algorithm thus follows the following rules :

1. If point B is closer to the extremity, then the segment intrudes (z-coordinate becomes positive).
2. If point A is closer to the extremity, then the segment protrudes (z-coordinate becomes negative).

Refer to Appendix C for a visual representation of an example.

4.6 Flat 3D Shape

In order to obtain perform the 'bending' of segments of the shape's skeleton, we need a 3D shape that had no curvature variations i.e. that is flat, which we will then 'bend' accordingly. To better visualize the concept of a 'flat' 3D shape,

imagine taking as input the 2D non-photo-realistic image, without any strokes, and rendering it as a 3D shape. In consequence, it has a null z-value and is in the plane. This will thus be the base of the next step. In addition, it also allows our algorithm to display this "flat" shape if the user decides not to sketch any strokes.

To implement this, on another canvas, we project the extremity points of the protruding/intruding segments of the 3D generated shape onto the plane thanks to the *Three.js Plane* class, a 2D surface that extends in 3d space, represented by a unit length normal vector and a constant. Hence, the normal vector takes as value the negative direction of the camera, and the constant point is any random point of the stroke. We then obtain an *intersect* line function in the *Plane* class, that takes as input the line defined by the center of the camera and the 3d constant point. This then gives us the intersection between the other points of the skeleton that aren't in the plane (i.e. that are on the protruding/intruding segments), and the plane. This method thus implements a projection from the camera point of view to 2D, giving us 3d points that are exactly on the plane, in a 'flat' 3D shape.

4.7 Bending of Flat 3D Shape

Once we have a 'flat' 3D shape (i.e. with no depth), we want to move a 3D point, using the same perspective camera to draw on the image plane and on the 3D object.

In 3D, we need to shift the extremities of the segments to bend them. This implies that, before the projection, we multiply the coordinates by the *View* matrix (i.e. attached to the perspective camera). Hence we convert the *World* coordinates to the *View* coordinates, and obtain a z-value. To be clear, the world matrix translates the coordinates of your vertices from model space to world space, whereas the view matrix translates those vertices from world space to camera/view space. This means the transformed coordinates are in relation to the camera/point of view.

Finally, depending on the result of the direction of curvature function (intruding or protruding), we can bend the two non-central segments forward (with positive z-value) or backward (with negative z-value) by approximately 45 degrees, an arbitrary value to clearly visualize bending. This is done by taking the length of the the segment we want to bend, and multiplying it by $sin(\pi/4)$ which is the new z-coordinate of the extremity. In the end, the x and y values of the ends of the 3D shape stay the same while the depth (or z values) are changed accordingly. Refer to Appendix B to see a visual of what occurs when bending.

5 Initial Ideas and their Drawbacks

The final algorithm described in 4 may seem quite simple, but in reality it was very demanding. To illustrate our 'journey' and the paths we took, we are going to discuss, in this section, the challenges we faced and solutions we explored before finding the right ones.

5.1 Skeleton Display

We initially thought of displaying the shape's skeleton through Three.js' skeleton constructor, which takes an array of bones and creates a skeleton that is used by a SkinnedMesh. However, the skeleton's geometry required information such as *skinIndices* and *skinWeights*, which weren't given with the final computation of the mesh's geometry. We thus went with the *THREE.lines* constructor.

5.2 Scaling and Drawing Strokes

Before thinking about scaling the camera from 3D to 2D as we did with the orthographic camera, we tried keeping the perspective cameras and drawing in 3D. We quickly realized that we wouldn't be able to manipulate the values since the scaling was off. We then decided to keep the three-dimensional scene and perform ray-tracing when it came to drawing curvature strokes. Ray-tracing allows you to create a vector from a 3D point in the scene, and detect which object(s) the vector intersects. However, after spending quite some time trying to implement this technique by adapting it to the existing Blobtree library, we noticed that there was a much simpler method : Three.js' orthographic camera.

5.3 Targeted Segment

We had two ideas to identify the targeted segment.

1. The first was to compute the Euclidean distance between the both segments' extremities and the center of curvature of the stroke. The endpoint that is the closest to the center of curvature is the one whose z coordinate will be changed. We then realized that instead of unnecessarily computing the center of curvature, we can

approximate it by simply taking the middle point of the two endpoints of the curve. Using the center of curvature can be considered as a possible extension. We also noticed that since we only had a list of segments, but didn't know which vertices correspond to which segment. We thus decided to color each segment of the skeleton to be able to make the distinction and find the targeted segment.

2. The second method was unnecessarily more complicated and required further research. The idea was to compute the potential field at the center of curvature and at each segment's extremity. Then, depending on the point with the highest value, we would pick the targeted extremity/segment.

5.4 Direction of Curvature

We had two initial ideas for computing the direction of curvature :

1. Similar to the current method, but with the center of curvature instead of the middle point between the two endpoints of the curve. We decided that we didn't need such a detailed computation, and that our middle point represented a simpler and sufficient approximation of the center of curvature.
2. Similar to the current method, but with the center of mass instead of extremities of the shape. We would go about the same logic, but swapping the points in both of our rules in section 4.5. Using the center of mass, although already computed in the shape generator file, only works in certain cases, and won't be useful in future applications.

6 Results

This report describes in detail the design of an interactive tool that allows the user to sketch curvature strokes on a 2D shape and visualize the 3D rendering.

The purpose of asking the user to recreate the generated 3D shape by sketching strokes on its 2D rendered image was to evaluate our classification methodology of curvature strokes. The goal was to find out if our user study conclusions are validated based on the user's strokes to acquire the desired result (assuming that the user follows our rules).

7 Conclusion

In conclusion, we have learned and understood the main concepts of Computer Graphics (such as skeletons, implicit surfaces, scaling cameras, ray-tracing...) and we know how to manipulate them through 3D shapes. We have become familiar with WebGL and its libraries such as Three.js, and have increased our fluency in Javascript. This is also our first research project, and it was the first time we were faced with an open-ended question that had such complexity. We had to redefine our project numerous times and simplify each step in order to efficiently make progress. We realized that a big aspect of research is trial and error as we learned and pivoted from the mistakes we made.

7.1 Future work

In the future, we could extend our algorithm to handle tubular shapes of higher genus and allow several sets of strokes on a same segment, given the segment is long enough. We could also eventually interpret the amount of curvature inferred by a given set of strokes, although it would require a large amount of research and artistic knowledge. Furthermore, we could compute the center of curvature if we eventually want more precision and avoid error with corner cases, as discussed in sections 5.3 and 5.4.

Finally, we could add features to the interface to enhance the UI, such as adding a pen icon to select before drawing and an eraser option.

Regarding the perceptual user study, our questionnaires gave us definitive responses, but it would be interesting to have people with an artistic background fill out our survey as they have more knowledge about sketching conventions.

Acknowledgments

We would like to thank our supervisors Marie-Paule Cani and Renaud Chabrier, and PhD student Tara Butler, who have helped us undertake this research. We are very thankful for their support, encouragement and insights throughout the project. Our meetings and conversations were vital in inspiring us to think outside the box, from multiple perspectives, to form a comprehensive and objective critique.

References

[1] GUY A. WYVILL B., GALIN E. The blob tree, warping, blending and boolean operations in an implicit surface modeling system. *The Visual Computer Journal*, 1997.

[2] Maxime Quiblier Marie-Paule Cani Cédric Zanni, Adrien Bernhardt. Scale-invariant integral surfaces. *Computer Graphics Forum, Wiley, pp.219-232*, 2013.

[3] Marie-Paule Cani-Loic Barthe Adrien Bernhardt, Adeline Pihuit. Matisse : Painting 2d regions for modeling free-form shapes. *https://hal.inria.fr/inria-00336688v1* , 2010.

[4] Renaud Chabrier. La géométrie de la vie: quand le dessin rencontre les sciences du vivant. *PhD thesis, Paris-Sciences-Lettres University, 2020*, 2020.

[5] Renaud Chabrier. La spatialité du trait: un processus d'interprétation basé sur les textures. *Journée du Groupe de Travail en Modélisation Géométrique, Laboratoire Lorrain de Rechercche en Informatique et ses application*, 2020.

Appendix

A First User Study

Hypothesis: People interpret curved strokes in a sketch as an indication of protrusion.

Initially, we needed to understand how people interpreted curved lines within drawings, more specifically whether they viewed 3D models as protruding or intruding when the lines are added. To do this we designed a questionnaire with 8 identical circles with curved lines inside, indicating curvature. Each circle was unique in that its lines were placed on different sides of the shape (up, down, left, right), and were curving in different directions.

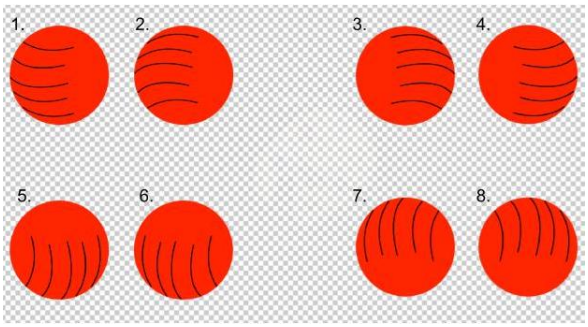

Figure 8: Sketches used in our first user study

In the questionnaire, each of these drawings were labeled 1 to 8, but ordered randomly. For every drawing, subjects were asked to state whether they saw the 3D drawing as protruding (popping out of the page towards them), or intruding (caving in to the page away from them). They only had these two options and could not leave any questions blank. They were also told that there is no "correct" answer.

A.0.1 Results

Some quick numbers on the subjects in our user study:

- Participants: 126
- Gender: 54% male, 46% female
- Age: 17% ($\leq$18), 72% (19-29), 11% ($\geq$30)

Below is a table with the outcomes of our first questionnaire.

Model Number	Percent that Perceived Protrusion	Percent that Perceived Intrusion
1.	82.4%	17.6%
2.	77.8%	22.2%
3.	70.4%	29.6%
4.	77%	23%
5.	74.6%	25.4%
6.	72%	28%
7.	72%	28%
8.	83.1%	16.9%

Figure 9: Results of our first user study

For every drawing, at least 70% of subjects perceived the model to be protruding, and many subjects believed every drawing was protruding.

B

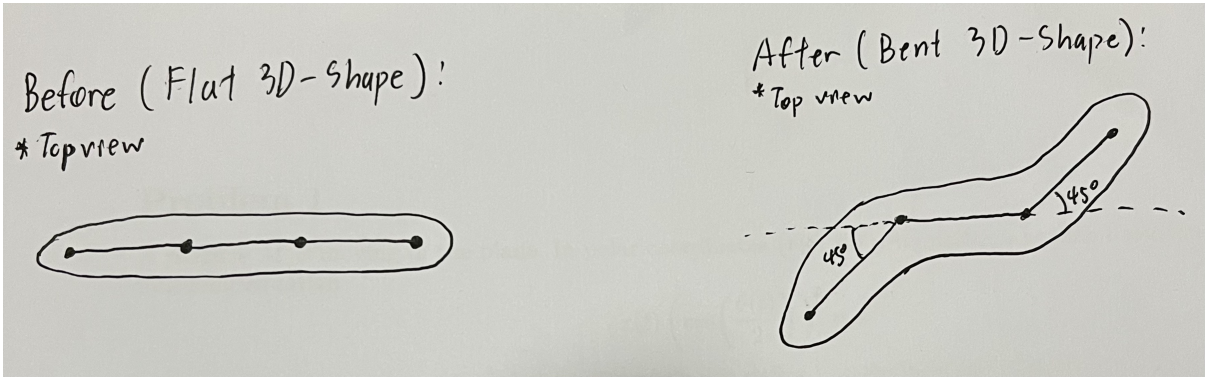

Figure 10: Flat to bent 3D shape

C

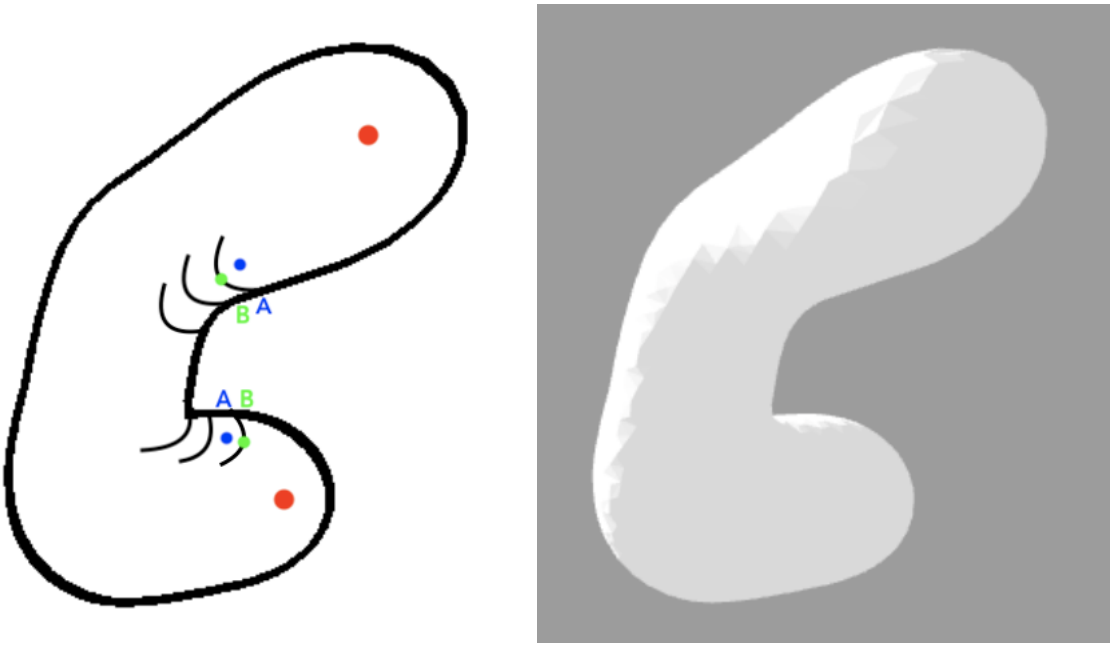

Figure 11: 2D image with strokes (left) to 3D shape (right)

Here, we can clearly see that the upper segment of the shape is protruding (pointing towards the user), whereas the bottom segment is intruding (pointing away from the user).