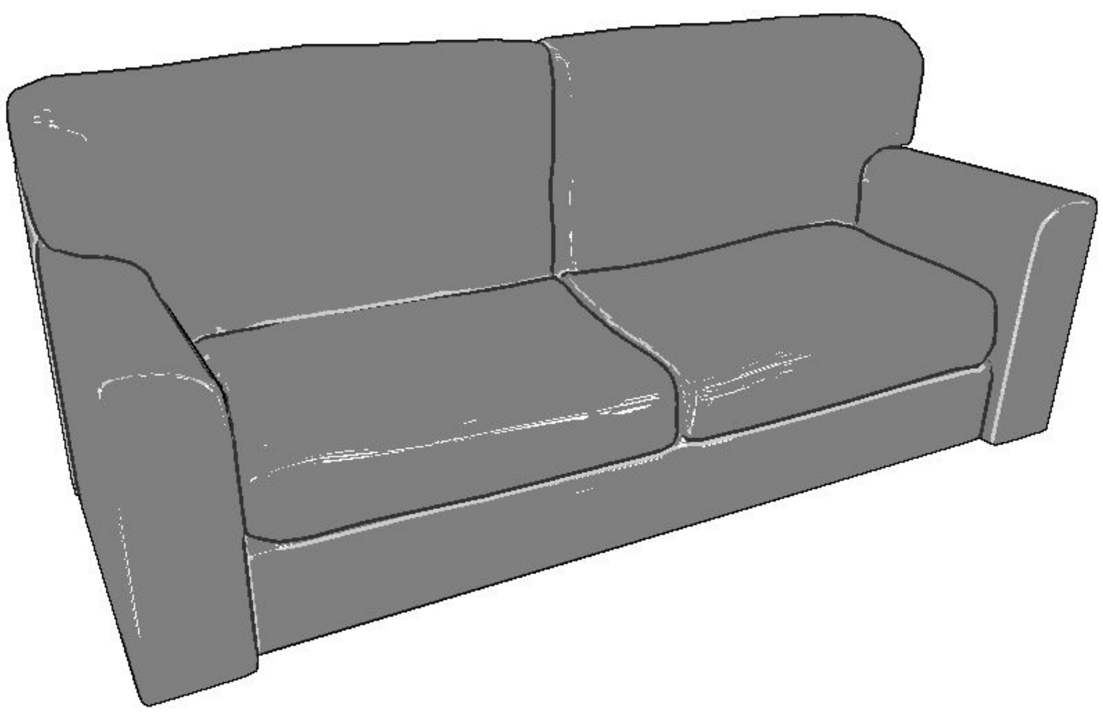
Lines for Conveying Shape

Beñat Morisset de Pérdigo

15th of December 2020, DigiPen Institute of Technology, Bilbao



0. Table of Contents

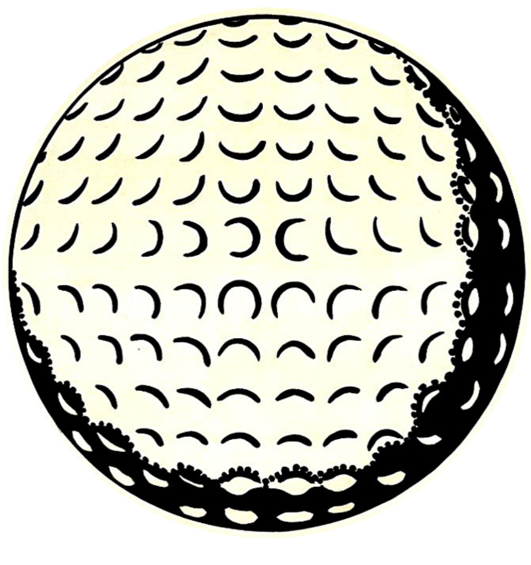
0. Table of Contents **2**
1. Introduction **3**
2. Goal **3**
3. Conventions **4**
4. Theory **4**
4.1 Principal Contour 5
4.2 Suggestive Contour 6
4.3 Principal Highlight 7
4.3.1 Finding the Axis of Maximum Curvature in Image-space 8
4.4 Suggestive Highlight 9
5. Implementation **10**
5.1 Principal Contour 11
5.1.1 Naive Approach 11
5.1.2 Sobel Filter of Z-Coordinate in Camera Space 13
5.2 Suggestive Contour 15
5.3 Principal Highlight 17
5.4 Suggestive Highlight 18
6. Results **19**
6.1 Style 21
6.2 Performance 22
6.2.1 Square Neighborhood Dimensions 23
6.2.2 Renderbuffer Resolution 23
6.2.3 Screen Coverage 24
7. Possible Improvements **25**
7.1 Appeal 25
7.2 Performance 26
8. Conclusion **27**
9. How to Use the Demo **27**
10. Bonus Images **31**
11. References **35**

1. Introduction

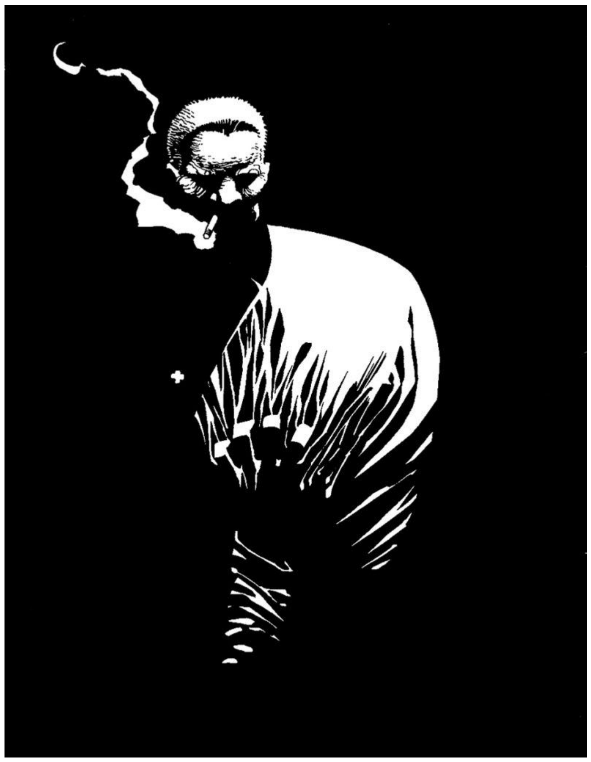
This document compiles the theory and implementation details of my final project for the course CS 562: Advanced Graphics that I have attended at Digipen Institute of Technology Europe - Bilbao during the semester of fall 2020. The implementation has been done in my real-time graphics framework for the course which also contains all the assignments and some bonus effects in other scenes.

2. Goal

My goal in this final project was to implement a non-photorealistic rendering technique in order to effectively convey 3D shapes using lines in a similar way an artist may do it. I tried to replicate the styles of Roy Lichteinstein's golf ball as well as Frank Miller's style.



Roy Lichtenstein, "Golf Ball", 1962
© Estate of Roy Lichtenstein



Sin City - Marv by Frank Miller

3. Conventions

For this document I consider the following conventions:

- The view vector v to be the unit vector that goes from the point in the surface to the camera
- The surface normal n to be the unit vector representing the surface normal pointing outwards of the object

4. Theory

The whole project is based on the following papers:

- Suggestive contours for conveying shape
- Highlight lines for conveying shape

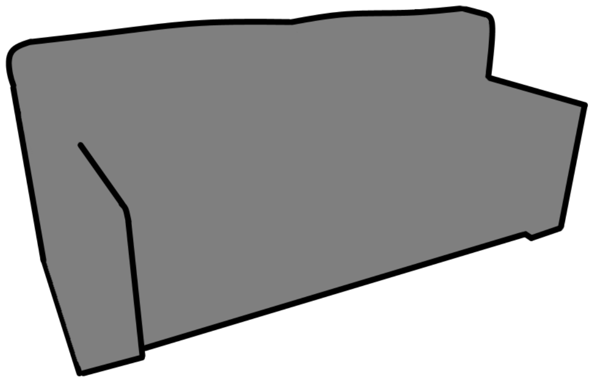
Those papers are closely related, and define a set of lines useful to convey 3D shape in a similar way an artist may do it. The lines defined in those papers are the following:

- Principal Contour
- Suggestive Contour
- Principal Highlight
- Suggestive Highlight

My purpose in this section is not to supplant the papers I based my project upon. I will give an informal summarized explanation of the core concepts that were needed to implement this effect.

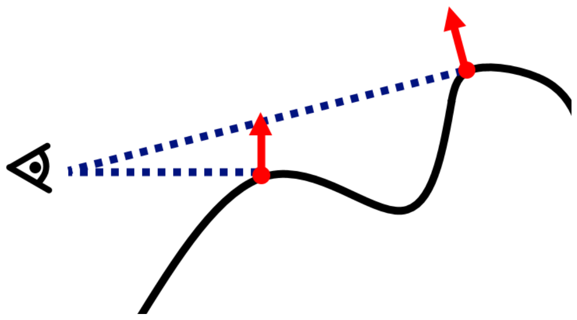
4.1 Principal Contour

The principal contour is a line that separates a front facing surface with a back facing surface (with respect to the viewer). It outlines the whole object and also marks the limits of self occluding features. On the image below you can see how the outline and the occluding armrest have principal contours.



This image made in paint.net represents schematically where the principal contour should be drawn

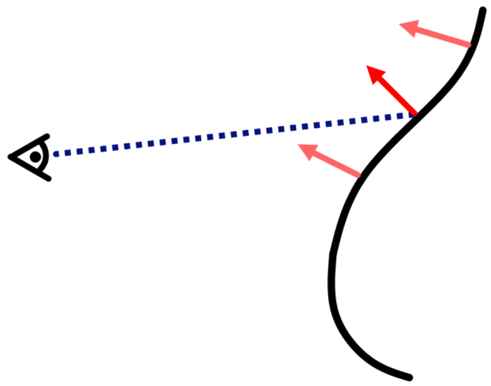
The paper suggests to draw principal contours on areas where the surface normal n is perpendicular to the view direction v. n is perpendicular to v when $n \cdot v = 0$.

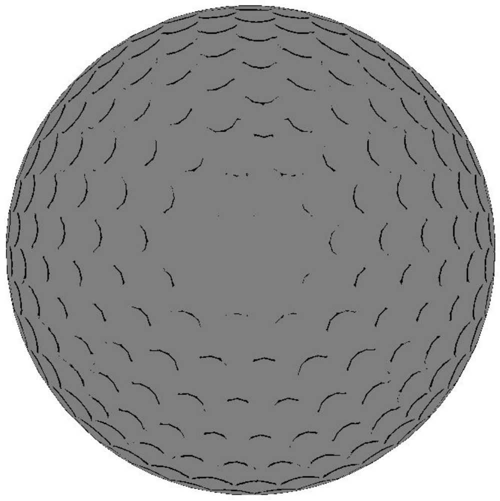


This image made in paint.net shows that contours are drawn on places where n is perpendicular to v

4.2 Suggestive Contour

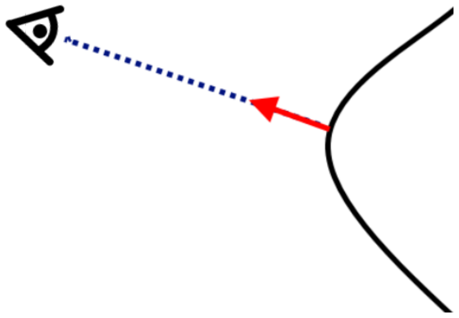
Suggestive contours are lines that give the illusion of indentations or concavities on a surface. Of course, if an indentation is deep enough, its ridge will occlude the deepest of it, and so a principal contour should be drawn instead. Therefore, a suggestive contour can be seen as an anticipation of a principal contour. The paper suggests to draw suggestive contours on local minima of $n \cdot v$.


A representation of the local minimum of $n \cdot v$

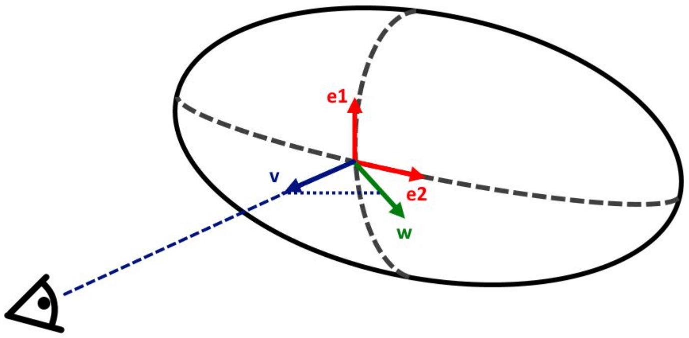

In this image taken from my framework we can see how the suggestive contours convey the concavities on the golf ball

4.3 Principal Highlight

Principal Highlights delineate surfaces that are directly facing the viewer. Therefore, they are on areas where n is parallel to v, or $n \cdot v = 1$. Another condition should be met to ensure the shape is a line: the projected view vector onto the surface's tangent plane w must be parallel to the axis of minimum curvature of the surface e_2. As e_1 is always perpendicular to e_2 it is equivalent to check that w is perpendicular to e_1.



Represents a point where n is parallel to v



This drawing represents how w relates to v, and e_1 and e_2 relate to the surface

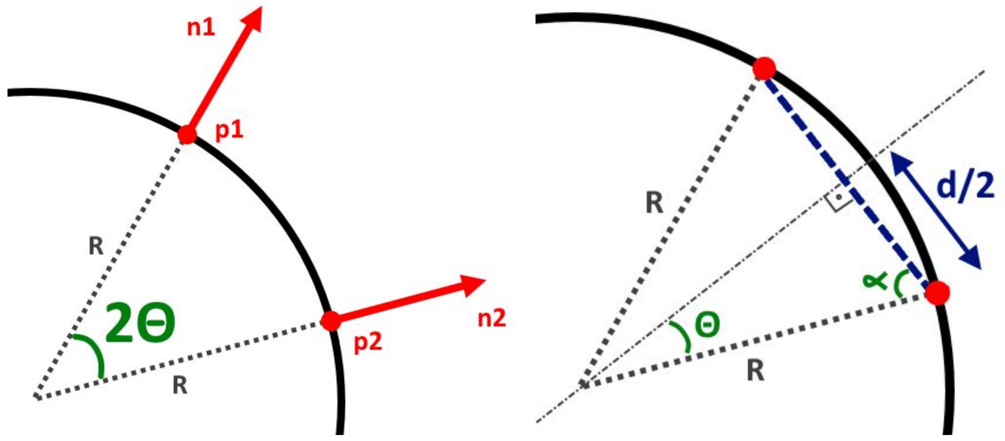
$w = \frac{v - (n \cdot v)\, n}{|v - (n \cdot v)\, n|}$ but to get e_1 the paper only details an object space implementation. So I came up with my own method to compute it in image space.

4.3.1 Finding the Axis of Maximum Curvature in Image-space

In 2D the curvature can be extracted from a pair of positions and a pair of normals. This also works in 3D if both points and both normals are contained in a normal plane.

The idea is to compute the curvature for each point in the neighborhood using the center-point and the center surface normal as the other pair, and find the maximum. The neighbor normal should be projected onto the center point's normal plane that goes through the neighbor point. Then the maximum axis of curvature is the direction from the center-point to the maximum curvature point, projected onto the tangent plane of the center point.

Let us go through the derivation of the formula to get the curvature C from two positions p_1, p_2 and two normals n_1, n_2, all contained in a normal plane:



Two drawings made on paint.net representing the relationship between the radius of curvature and the two pairs of point and normal in the normal plane.

As you can see in the drawings below, with a bit of trigonometry we find that we can compute the radius of curvature R as $R = \frac{d}{2 \cdot sin(\Theta)}$ or $R = \frac{d}{2 \cdot cos(90^{o} - \alpha)}$, and $\Theta = \frac{arccos(n_1 \cdot n_2)}{2}$, $\alpha = 90^{o} - \Theta$

Finally $C = 1/R$.

This last step can be skipped as we do not care for the exact value of the curvature, we just need to find its local maxima. For that it suffices to find the local minima of R as C is inversely proportional to R.

The projection of the neighbor normal into the normal plane of the center point going through the neighbor point is:

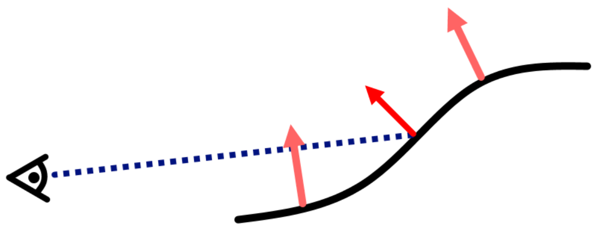
$m = \frac{n_n - (n_n \cdot v) \cdot v}{|n_n - (n_n \cdot v) \cdot v|}$ with $v = n_c \times (p_n - p_c)$ where

- p_c , p_n are the positions of the center and the neighbor respectively
- n_c , n_n are the normals of the center and the neighbor respectively
- m is the resulting projection of n_n

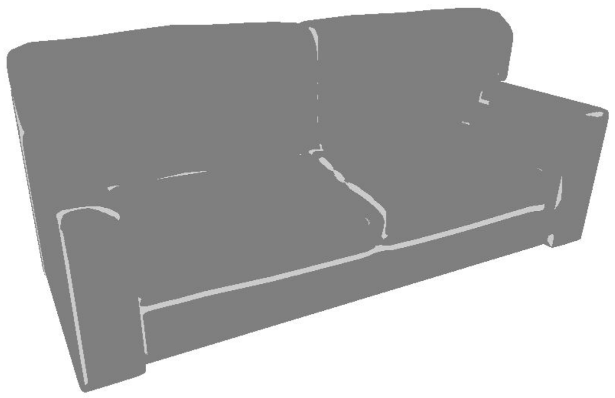
4.4 Suggestive Highlight

Suggestive highlights are lines that give the illusion of bumps or bulges on a surface. Of course, if a smooth bump is high enough, it will have a part that is directly facing the viewer, so a principal highlight should be drawn instead. Therefore, a suggestive highlight can be seen as an anticipation of a principal highlight.

The paper suggests to draw suggestive contours on local minima of $n \cdot v$.



A representation of the local maximum of $n \cdot v$



In this image taken from my framework we can see how the suggestive highlight convey the convexities on the sofa

5. Implementation

The framework uses OpenGL API and the following libraries: GLFW, glad, glm, stb_image, assimp, JsonCpp and Dear ImGui. The rendering is done with the deferred shading technique.

The whole effect is done in a single fragment shader that uses the normal and depth buffers of the g-buffer. To be able to showcase the effect better, I gave the shader many different options and tweakable variables, but when used for a practical application, it is better to strip the effect of all unneeded options to gain performance. Especially the options that let you choose between different algorithms like whether or not to use a circular neighborhood for the suggestive lines, or which principal contour method to use.

To avoid problems the first step in the shader is to discard all clear color pixels.

5.1 Principal Contour

5.1.1 Naive Approach

The first approach I implemented follows closely what the papers suggests. It tints pixels that have a surface normal n close to perpendicular to the view direction v.

I considered a n to be perpendicular to v when the dot product between those two unit vectors is smaller than a tweakable value. If using face normals, the result of the dot product cannot be negative as the gbuffer does not contain back faces. If using normal maps, $n \cdot v$ can be below zero, so we have to clamp the result to zero. This way we only get results between zero and the tweakable value.

To make the lines fade smoothly I added a fade margin in which the color is interpolated between the principal contour color and the base color. This is not needed when suggestive contours are present as they will naturally extend the principal contour.

This implementation gives the following artifacts:

- Flat surfaces that have n almost perpendicular v will be colored entirely as a contour. The tinted area is no longer a line and is, therefore, out of place in this effect.
- On surfaces that abruptly change from frontface to backface, so that no n close to perpendicular to v is captured in the g-buffer will not be tinted. This gives images where the principal contour is absent of many places where it should be present.

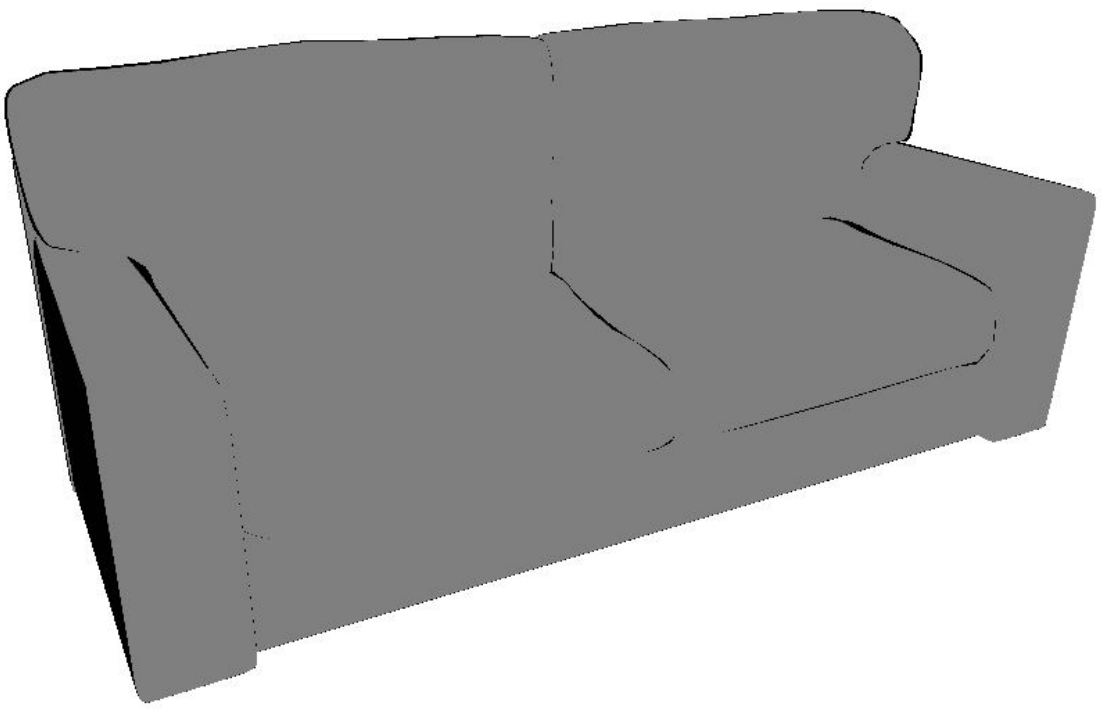


Image from my framework with only principal contours enabled. The principal contours are detected using the method explained above. The two artifact types are visible. On the left armrest panel we can see the almost perpendicular flat surface artifact. On the right armrest we can see the missing contour due to the abrupt change from frontface to backface artifact.

5.1.2 Sobel Filter of Z-Coordinate in Camera Space

This approach is farther from the papers' than the previous one as it does not take into consideration the surface normal n nor the view direction v.

This implementation comes from the hypothesis that principal contours are places where the camera-space z-coordinate z changes abruptly. In fact, n in such a place will be close to perpendicular to v. So, I will consider part of a principal contour any pixel that has a camera-space z-coordinate gradient G bigger than a tweakable value. The catch is that some places where the normal is close to perpendicular may not have a sufficiently abrupt change in the camera-space z-coordinate to be tinted. I did not find this to be critical for this effect.

To compute the G of a pixel I compute the z of all the pixels in a 3x3 neighborhood from their depth d and applying the Sobel operation to those.

To get z from d, I used the following formula: $z = \frac{q_{4,3}}{q_{3,4} \cdot (2 \cdot d - 1) + q_{4,4}}$ where $q_{i,j}$ is the element at row i and column j of the inverse of the projection matrix.

This equation is derived from the formula to get the camera-space position from the depth value d and the texture coordinates u and v: $pos_{cam-space} = \frac{1}{p_w} \cdot p$ with $p = P^{-1} \cdot pos_{clip-space}$ and $pos_{clip-space} = 2 \cdot \left(u - \frac{1}{2},\ v - \frac{1}{2},\ d - \frac{1}{2}\right)^T$.

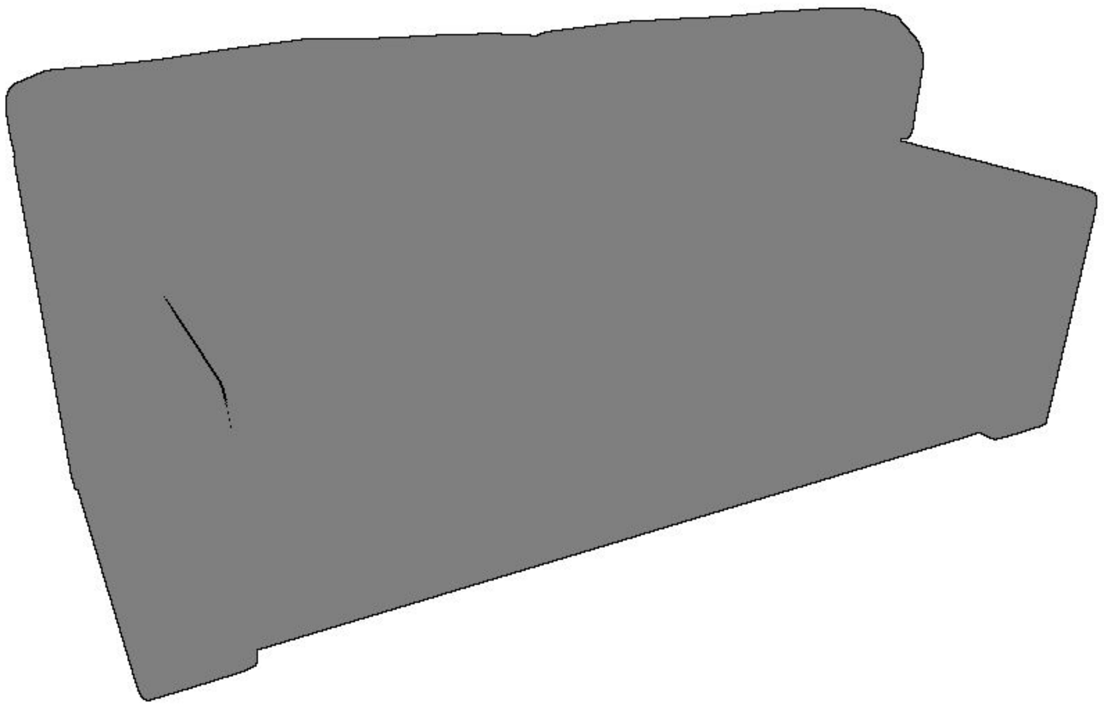


Image from my framework with only principal contours enabled. The principal contours are detected using the method explained above. The artifact is visible on the left armrest as the inner contour line does not outline the front part of the armrest as there the z-coordinate gradient is not big enough.

5.2 Suggestive Contour

To draw suggestive contours I mostly followed the algorithm described in the papers.
The suggestive contour should be drawn on local minima of $n \cdot v$. As a value of 0 will be a local (co-)minimum, this method also detects principal contours. Therefore I draw the principal contour after the suggestive contour to ensure the principal contour is drawn with the correct color.

To find those local minima of $n \cdot v$ I compute $n \cdot v$ on a NxN pixel neighborhood, where N is a tweakable integer. A simple version would be to consider a center pixel a local minimum if it has the smallest value in the neighborhood.

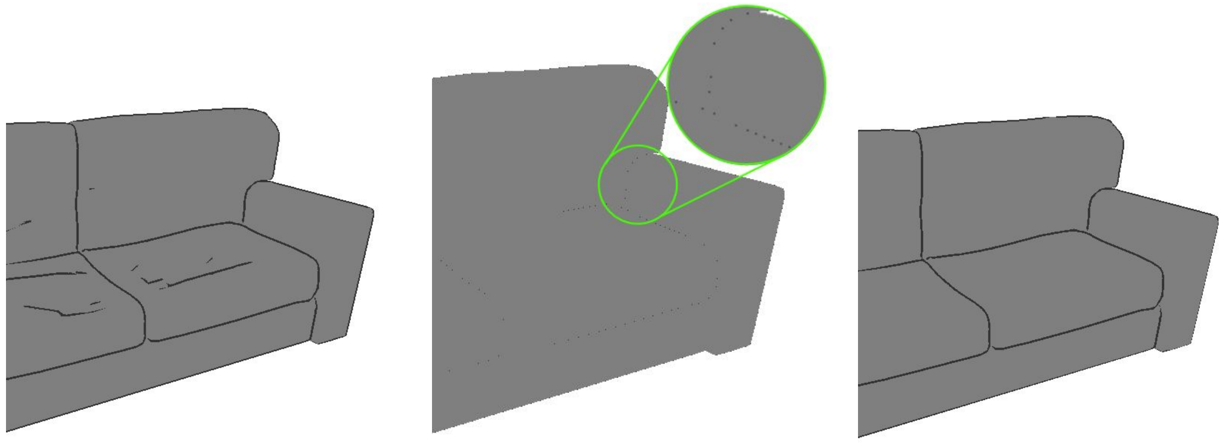
This simple version has two problems:

- Mostly flat surfaces will also get suggestive contours as this minimum does not consider by how much is the pixel minimum
- Suggestive contours will be dotted lines as if a pixel is a local minima in a NxN neighborhood it means that the next N pixels in a direction cannot be a minima.

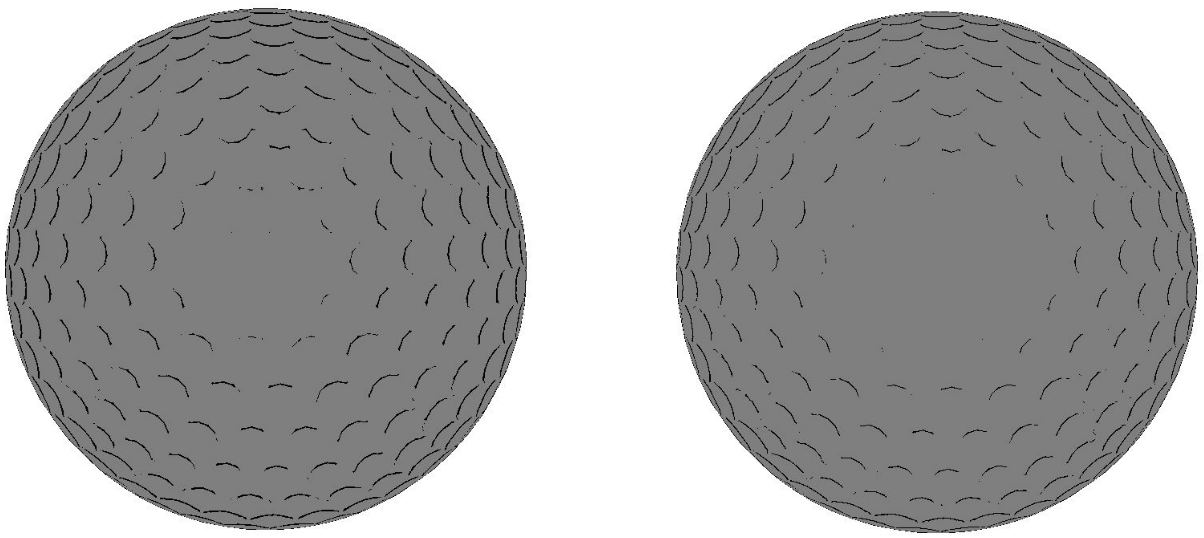
To solve this artifacts the paper suggests that to introduce two tweakable values:

- A minimum of difference between the center pixel and the maximum in the neighborhood. This way mostly flat surfaces will not get suggestive contours if we do not wish to.
- An acceptable proportion of pixels that can have a smaller value than the center one, and still consider the center a minimum. This way we can tweak the value until the lines are nice and continuous.

I also discard neighbors that are too far away in terms of the Z-coordinate in camera space to avoid counting pixels from non-neighboring surfaces as neighbors.


All three images show renders of suggestive contours in my framework. The left image lacks the minimum difference check. The center image lacks the acceptance of a small proportion of smaller values. The right image has both tweakable values.

Instead of a NxN square neighborhood the paper suggests to use a circular neighborhood of radius N. I implemented that version as well, just discarding pixels that are further than N away from the center pixel, and found out that it does not seem to improve on the effect.


Both images are taken from my framework. Left uses a square neighborhood to find the local minima of $n \cdot v$ *, the right image uses a circular neighborhood.*

5.3 Principal Highlight

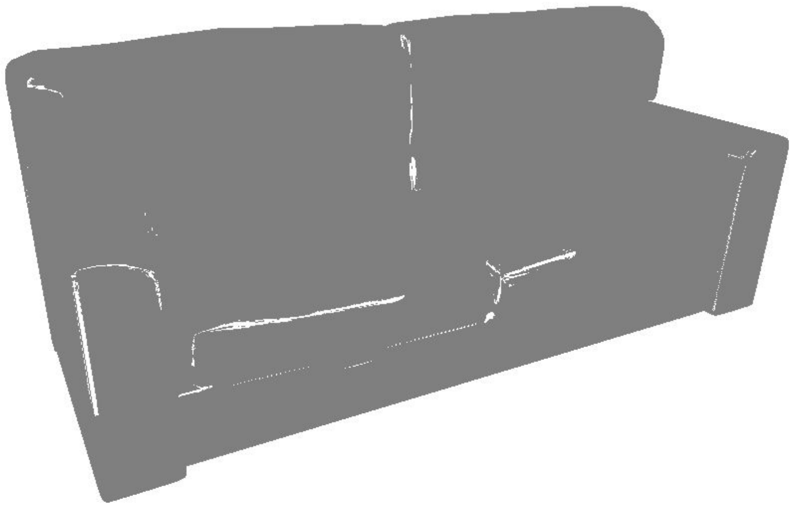
The paper suggests to consider as a principal highlight places that pass two conditions:

- Surface normal n almost parallel to the view direction v
- Axis of smallest curvature e_2 is almost parallel to the projected view direction w

For the first condition it suffices to check that $n \cdot v$ is bigger than a tweakable value. This is analogous to the naive implementation of the principal contour.

For the second condition, because the axis of maximum curvature e_1 is perpendicular to e_2, it is enough to check that $|e_1 \cdot w|$ is smaller than a tweakable value.
$w = \frac{v - (n \cdot v)\, n}{|v - (n \cdot v)\, n|}$ and to get e_1 I proceeded as explained in the section 4.3, taking a square neighborhood NxN.

I found that the final projection into the tangent plane can be omitted and the result still looks almost the same.



This is an image taken from my framework of the principal highlights of a sofa.
It uses a 17x17 square neighborhood to find the axis of maximum curvature.

5.4 Suggestive Highlight

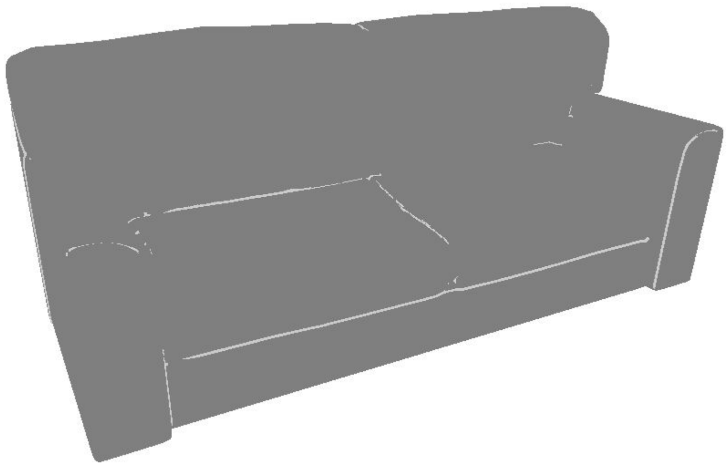
To draw suggestive highlights I mostly followed the algorithm described in the papers, which is the same as suggestive contours, but finding local maxima of $n \cdot v$ instead of local minima.

$n \cdot v = 1$ will be a local maximum, so this method also detects principal highlights. Therefore I draw the principal highlights after the suggestive highlights to ensure the principal highlights are drawn with the correct color.

To find those local maxima of $n \cdot v$ I compute $n \cdot v$ on a NxN pixel neighborhood, where N is a tweakable integer.

The same artifacts as for suggestive contours would appear with the naive implementation so we add the same tweakable values to correct them:

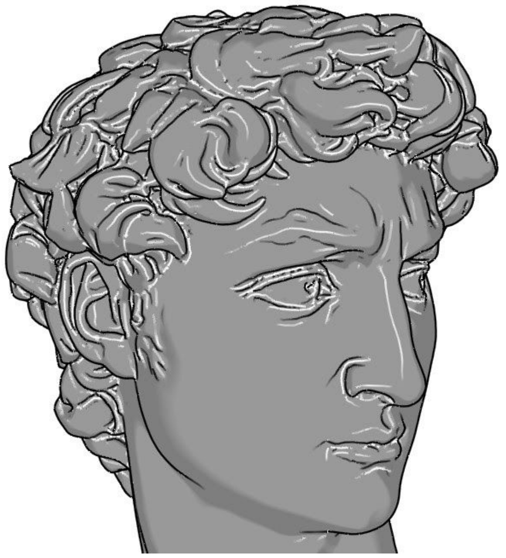
- A minimum of difference between the center pixel and the maximum in the neighborhood. This way mostly flat surfaces will not get colored if we do not wish to.
- An acceptable proportion of pixels that have a bigger value than the center one, and still consider the center a maximum. This way we can tweak the value until the lines are nice and continuous.



This is an image taken from my framework of the suggestive highlights of a sofa. It uses a 9x9 square neighborhood to find the local maximum.

6. Results

As you can see in the images below I achieved an effect that is similar to the one done by the papers' authors. As expected the papers's object-scpace algorithm gives smoother lines and therefore a more appealing result.


Left image of figure 1 of the paper "Highlight Lines for Conveying Shape". It has been done in object-space.

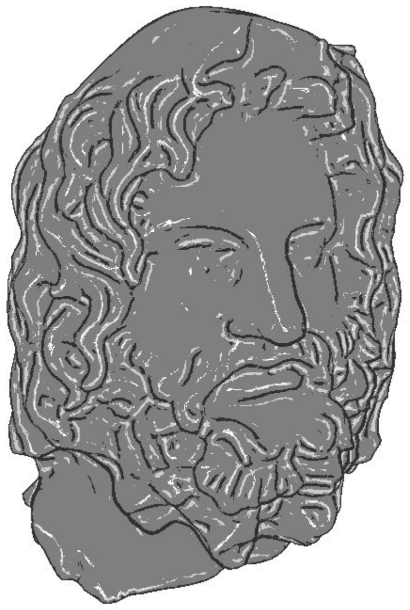


Image taken from my framework. It lacks the subtle toon shading the paper's has

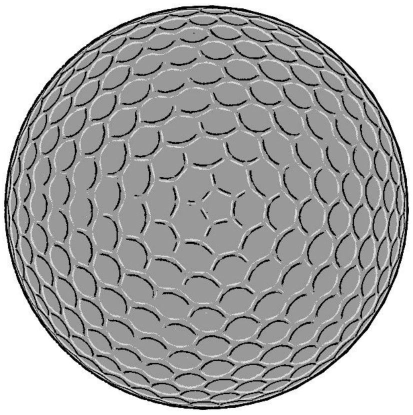


Image-space render of figure 12 of the paper "Highlight Lines for Conveying Shape"

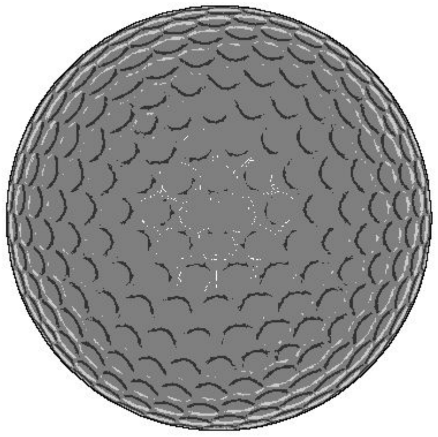


Image taken from my framework.

I noticed the effect deteriorates when decreasing the vertex count.

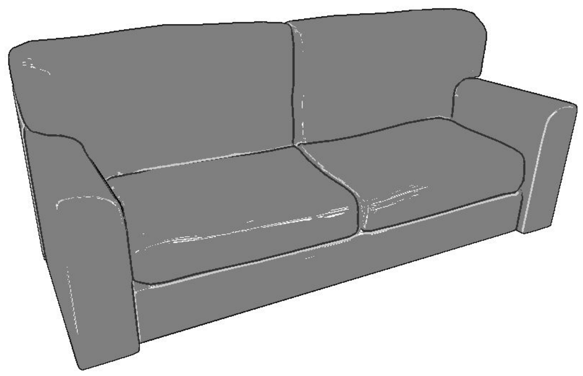


Image taken from my framework of the sofa with 3238 vertices

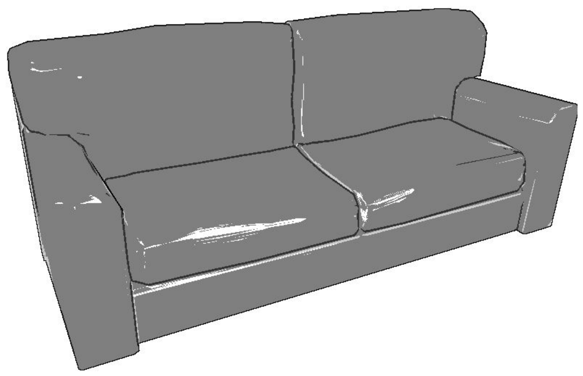
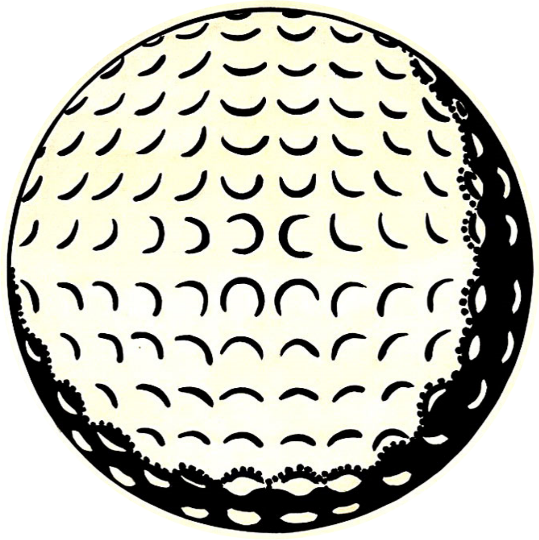


Image taken from my framework of the sofa with 412 vertices

Although my framework supports normal mapping, the objects I used for this project do not have normal maps. I expect the effect would improve with proper normal maps as it makes normal transitions smoother.

6.1 Style

It requires a lot of manual tweaking of the values to get the desired look, I have done my best to replicate Frank Miller and Roy Lichstenstein. You can see the results below.


Roy Lichtenstein, "Golf Ball", 1962
© Estate of Roy Lichtenstein

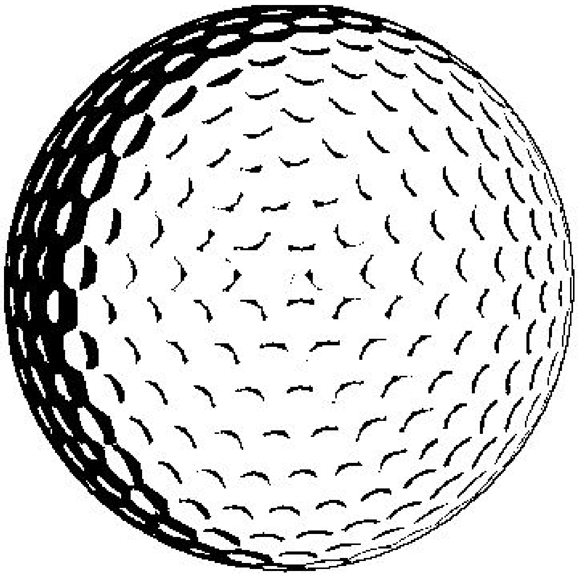
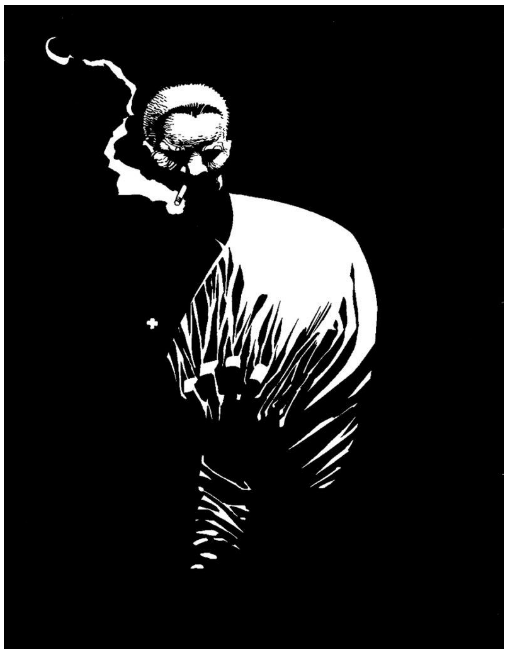


Image taken from my framework


Sin City - Marv by Frank Miller

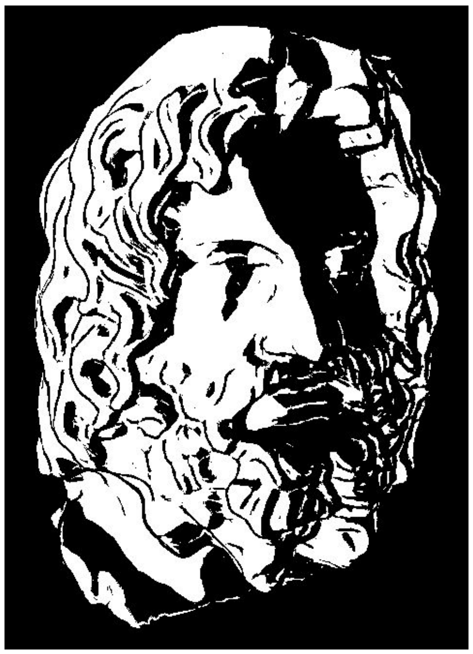


Image taken from my framework

6.2 Performance

This project was not done with performance in mind, so the implementation is probably very sub-optimal. Nonetheless, I have done some little benchmarks of the effect so you can get a rough idea of what to expect. I added a shader with less options and one with no options as a clumsy attempt at optimization.

The shader with less options does not have:

- Choice of primary contour type (uses the sobel of camera z coordinate)
- Choice of neighborhood type (uses square neighborhood)
- Choice of turning on and off individual line types (all are drawn)
- Choice of scaling the allowed proportions and minimum dot difference for suggestive lines by the distance between the neighbor and the center pixel (no scaling is done)
- Choice of turning on and off the toon-shading (always has it on)

This should be a nice improvement as it removes a lot of branching in the shader. This should especially be important in the choices of algorithms, as there both branches have potentially heavy computations.

On the shader with no options only two uniforms remain: the view and the projection matrix. Both are passed to the GPU every frame so an improvement would be to pass the projection matrix only when it changes, which is in most applications very rare. Compared with the less options shader, this only removes tweakable values. So it should have a smaller impact.

All the benchmarks below have been done in a machine with processor Intel® Core™ i7-7700 CPU @ 3.60GHz and graphics card NVIDIA GeForce GTX 1050.

6.2.1 Square Neighborhood Dimensions

As you can see below, the performance of the effect is highly dependent on the dimensions used for the neighborhoods. This is expected as there are three double loops that depend on N so the algorithm is $O(N^2)$.

As expected, removing a few branching options makes a noticeable difference. Removing the rest of the tweakable values also seems to have a small impact.

	square neighborhood dimension		
	5x5	**9x9**	**13x13**
shader with all options	5.6 ms	18.9 ms	39.5 ms
shader with less options	5.2 ms	17.7 ms	38.1 ms
shader no options	-	16.6 ms	-

Benchmark done with a 1600x900 renderbuffer resolution with the effect covering it fully.

6.2.2 Renderbuffer Resolution

As all effects based on a fragment shader, the performance cost is directly proportional to the renderbuffer resolution, as the full effect is done once per pixel.

Again, the less options shader seems to be a noticeable improvement and the no options shader yields a smaller improvement, but is still noticeable.

	renderbuffer resolution		
	800x450	**1200x675**	**1600x900**
shader with all options	3.6 ms	10.2 ms	18.9 ms
shader with less options	3.2 ms	9.3 ms	17.7 ms
shader no options	3.0 ms	9.1 ms	16.6 ms

Benchmark done with all 9x9 square neighborhoods with the effect covering the renderbuffer fully.

6.2.3 Screen Coverage

All my shaders discard clear color pixels to avoid problems. This is, as a side effect, an optimization. Therefore, the full effect is only applied to the pixels belonging to an object, so the number of pixels that get the full effect is a very important factor.

	screen coverage proportion		
	full	**~1/4**	**~1/35**
shader with all options	18.9 ms	3.3 ms	0.7 ms
shader with less options	17.7 ms	3.2 ms	0.7 ms

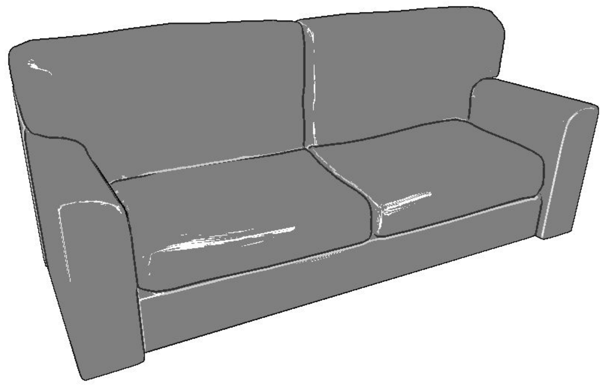
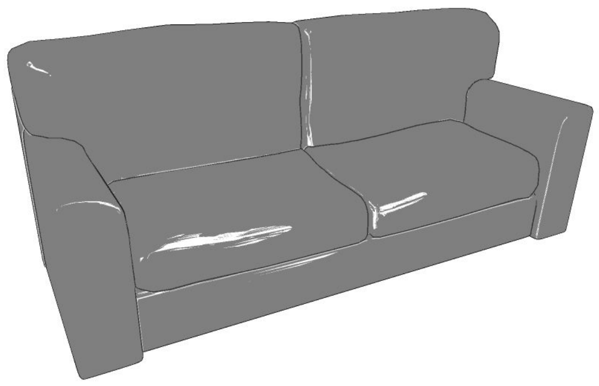
Benchmark done with all 9x9 square neighborhoods with a 1600x900 resolution.

I forgot to add the no-options shader and now it will be very hard to get the exact same coverage as the other two. So I leave it like this as the most important information is conveyed here and I do not want to redo the benchmark entirely again.

7. Possible Improvements

7.1 Appeal

I have implemented one improvement: Super-sampling Anti-aliasing (SSAA). This smoothes out the lines increasing the effectiveness of the effect. Unfortunately, this technique is quite expensive for this effect as, for Mx SSAA it multiplies the cost of the fragment shader by M^2 ! This makes the framework run way below the 60F PS mark in my computer which means it is unusable for most real-time applications without some heavy optimization. You can try it for yourself following the instructions in section 9.

These images have been taken from my framework. They show the same render, the left not having SSAA and the right one having 2x SSAAx2. As expected the rendering time has been multiplied by 4 (from 10.3 ms on the left one to 40.2 ms on the right one)

An easy improvement would be to fade the lines when they near their limit values. I have done this only for the Principal Contour, but the paper states that this is especially important on both suggestive lines.

The biggest improvement in appeal would come from an improvement of the Principal Highlight. As we have seen in section 5.3, my implementation yields badly defined Principal Highlight lines.

As stated in section 6, using proper normal maps should also improve the results.

And off course, having more detailed objects will yield better results. So prefer using high-poly meshes when using this effect.

Changing the approach completely and doing all in object-space as the papers' authors did would greatly improve the lines definition, but that would be a whole other project.

7.2 Performance

As it has already been stated, this project was not made with performance in mind. Therefore, this implementation can probably be optimized in many ways. Anyway, I list below a couple optimizations I thought about.

As we have seen, removing the unneeded options does optimize noticeably. As stated in the section 6.2 the projection matrix could also be sent only when it changes, which is often never the case.

If only using face normals, we could work only with the depth buffer, computing the normal from dFdX and dFdY of the position in camera-space via the tangent and the bitangent. This would enable the removal of the normal buffer form the g-buffer which should greatly optimize the geometry pass (which is filling the g-buffer) and also optimize the effect. Keep in mind that optimizing the geometry pass is only useful when dealing with a huge amount of vertices.

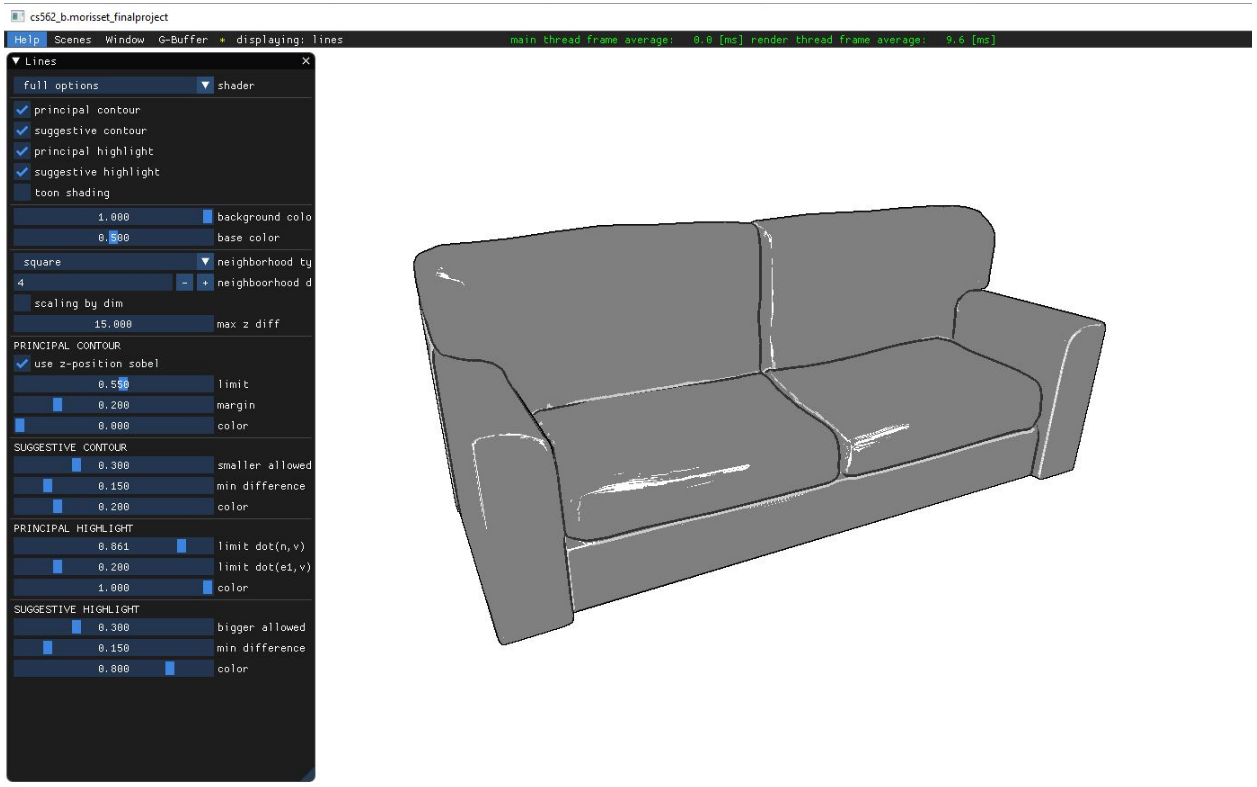
8. Conclusion

Although not perfect, this project has achieved its goal to render 3D objects using lines in a similar way an artist might. The downside is this implementation is a bit expensive and needs optimizing to be usable for a real-time application such as a game.

9. How to Use the Demo

Next to this document there should be a folder called BIN. Open it and execute the file called cs562_b.morisset_finalproject.exe. Inside the folder there is also a file called README.txt that contains a brief explanation of the controls, you can read it as well.

Once the demo has loaded (it should not take more than a few seconds) you will see the following on your window:



```
* displaying: lines          main thread frame average:   0.0 [ms] render thread frame average:   9.6 [ms]

full options              shader
[✓] principal contour
[✓] suggestive contour
[✓] principal highlight
[✓] suggestive highlight
[ ] toon shading
1.000                     background colo
0.500                     base color
square                    neighborhood ty
4                         neighboorhood d
[ ] scaling by dim
15.000                    max z diff
PRINCIPAL CONTOUR
[✓] use z-position sobel
0.550                     limit
0.200                     margin
0.000                     color
SUGGESTIVE CONTOUR
0.300                     smaller allowed
0.150                     min difference
0.200                     color
PRINCIPAL HIGHLIGHT
0.861                     limit dot(n,v)
0.200                     limit dot(e1,v)
1.000                     color
SUGGESTIVE HIGHLIGHT
0.300                     bigger allowed
0.150                     min difference
0.800                     color
```

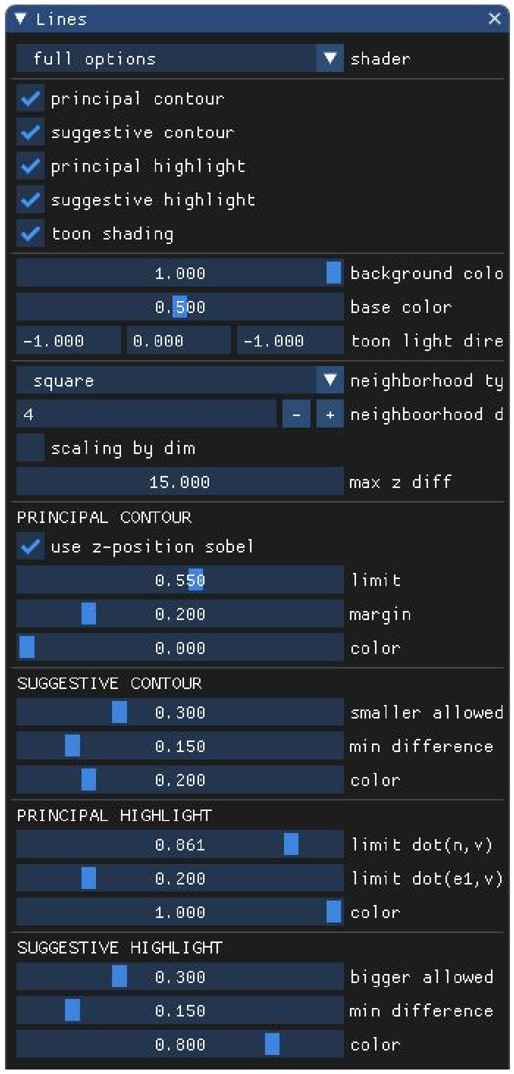
Framework window upon opening.

The top bar contains the following:

- **Help**: contains a brief explanation of the controls and shortcuts for the scene. Please read this as it contains information not listed below.
- **Scenes:** here you can reload the whole scene, just the shaders or change the current scene. Most of the scenes are for other assignments; final_project_sponza and final_project (the one you start in) are the two relevant scenes for this project.
- **Window:** here you can change the window resolution as well as open/close the Lines floating menu.
- **G-Buffer:** here you can change the g_buffer resolution separately from the window's. This is useful to get Super-Sampling Anti-Aliasing (SSAA).
- **Displaying:** It states what is currently being displayed. You can change it to see the normal buffer, the depth buffer, and the positions in camera-space. To see the effect it should be in lines. This is useful for debugging and understanding how the framework works.

The top bar also states how much time in milliseconds the framework is spending on computations in the CPU and in the GPU. As this framework is only used for graphics, the CPU is not used much. The numbers are displaying a moving average so when checking, leave it without touching anything for a few seconds for it to stabilize.

The lines floating window displays all the options and tweakable values of the Lines for Conveying Shape effect. Remember that if you have closed it you can reopen it in the Window menu on the top bar.


▼ Lines ×
full options ▼ shader
✓ principal contour
✓ suggestive contour
✓ principal highlight
✓ suggestive highlight
✓ toon shading
1.000 background colo
0.500 base color
-1.000 0.000 -1.000 toon light dire
square ▼ neighborhood ty
4 - + neighboorhood d
☐ scaling by dim
15.000 max z diff
PRINCIPAL CONTOUR
✓ use z-position sobel
0.550 limit
0.200 margin
0.000 color
SUGGESTIVE CONTOUR
0.300 smaller allowed
0.150 min difference
0.200 color
PRINCIPAL HIGHLIGHT
0.861 limit dot(n,v)
0.200 limit dot(e1,v)
1.000 color
SUGGESTIVE HIGHLIGHT
0.300 bigger allowed
0.150 min difference
0.800 color

This image taken from my framework show all the possible options and tweakables of the Lines floating menu.

The first dropdown menu lets us choose the shader to use. In section 6.2 I have explained what each does, essentially each next shader has less tweakables and options and should have better performance. Here I will explain all the options and tweakables of the "full options" hader, as it is the one containing the most and none of the others have things this does not.

The five checkboxes let you enable/disable each type of line separately as well as the toon shading.

The following two sliders let you choose the background and base greyscale colors.

The following vector value defines the light direction for the toon shading. Do not worry, the shader normalizes it before use.

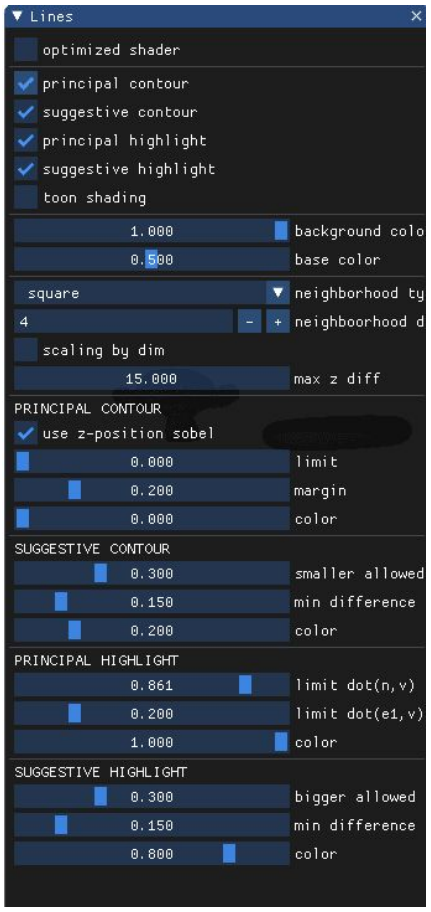
The following dropdown menu and integer input are the neighborhood type and dimension. See section 4.2, 4.3 and 4.4 for detailed explanations. Both apply to both suggestive lines as well as the Principal Highlight. The neighborhood radius R for circular neighborhood or the square side N for square neighborhood is going to be $R = N = 2 \cdot dim + 1$.

Scaling by dim is a functionality suggested by the paper which scales both tweakable values, the proportion allowed and the minimum difference of dot product, of both suggestive lines by N or R and by $\frac{1}{N}$ or $\frac{1}{R}$ respectively. This has not been explained above but I found out it is quite useless as values should be both values should be tweaked anyways.

Max Z diff is the difference in the camera-space Z-coordinate we accept for our neighbors as explained in section 4.2. Again, this is used in all linetypes but the Principal Contour.

Finally, each line type has its own set of tweakables and options. They are explained in their corresponding sections: 4.1, 4.2, 4.3 and 4.4.

10. Bonus Images



```
▼ Lines
[ ] optimized shader
[✓] principal contour
[✓] suggestive contour
[✓] principal highlight
[✓] suggestive highlight
[ ] toon shading
1.000        background colo
0.500        base color
square ▼     neighborhood ty
4      - +   neighboorhood d
[ ] scaling by dim
15.000       max z diff
PRINCIPAL CONTOUR
[✓] use z-position sobel
0.000        limit
0.200        margin
0.000        color
SUGGESTIVE CONTOUR
0.300        smaller allowed
0.150        min difference
0.200        color
PRINCIPAL HIGHLIGHT
0.861        limit dot(n,v)
0.200        limit dot(e1,v)
1.000        color
SUGGESTIVE HIGHLIGHT
0.300        bigger allowed
0.150        min difference
0.800        color
```

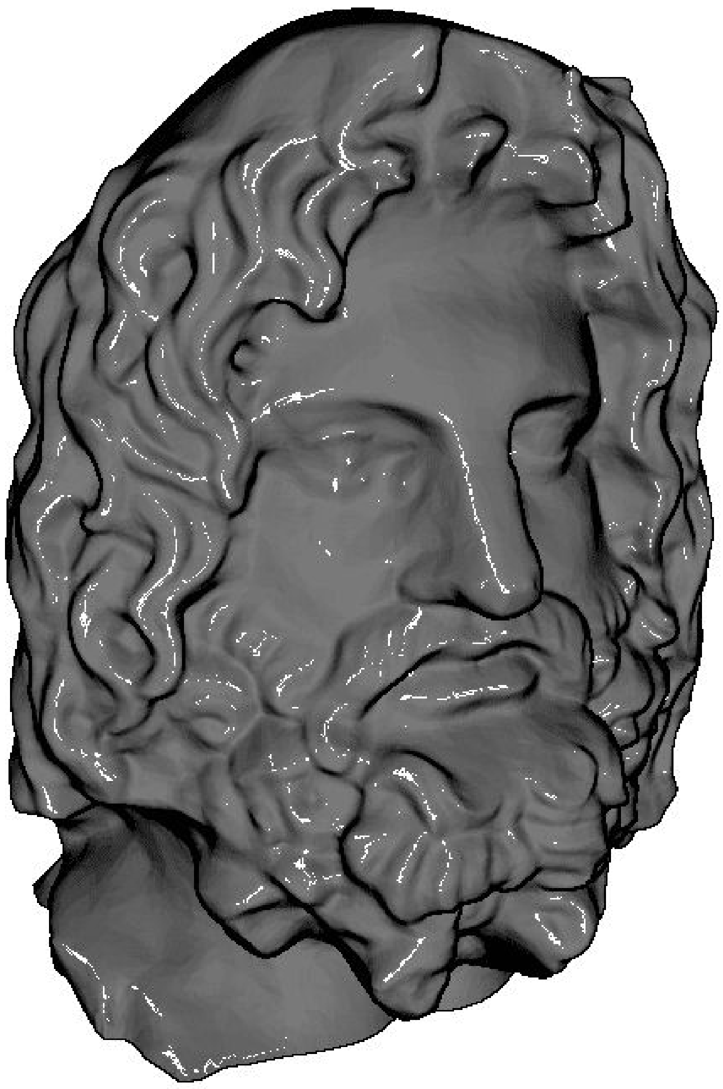
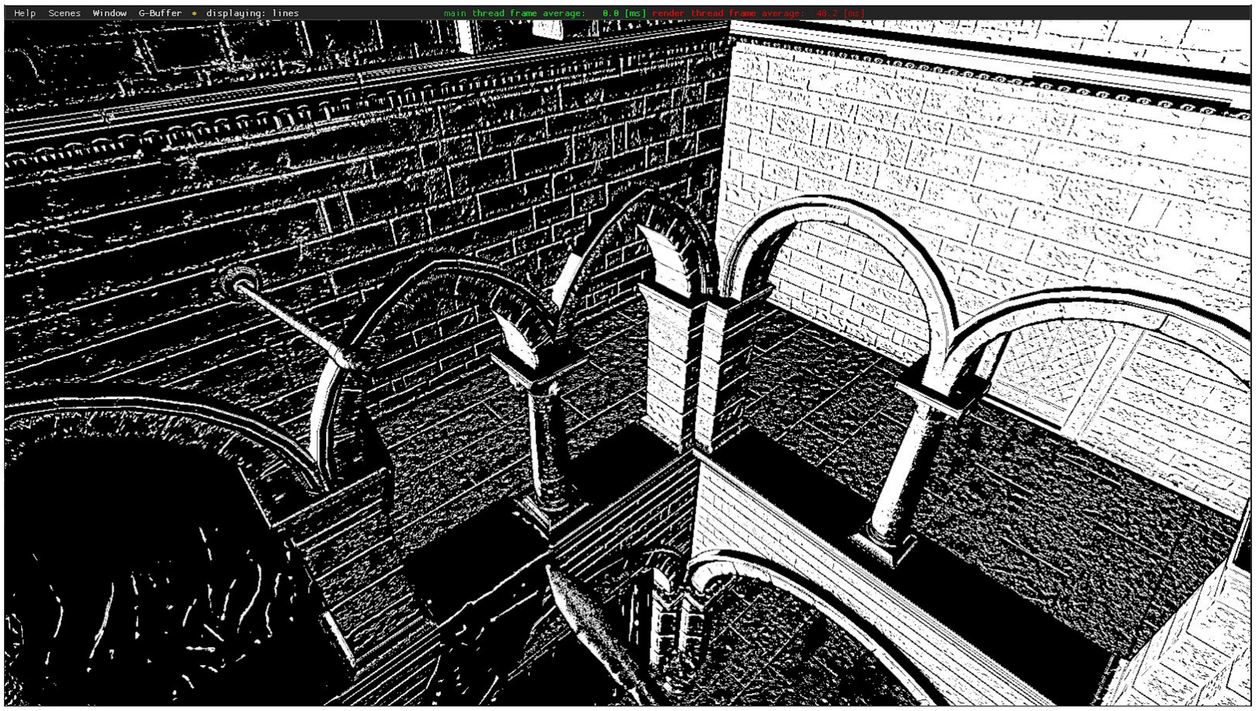
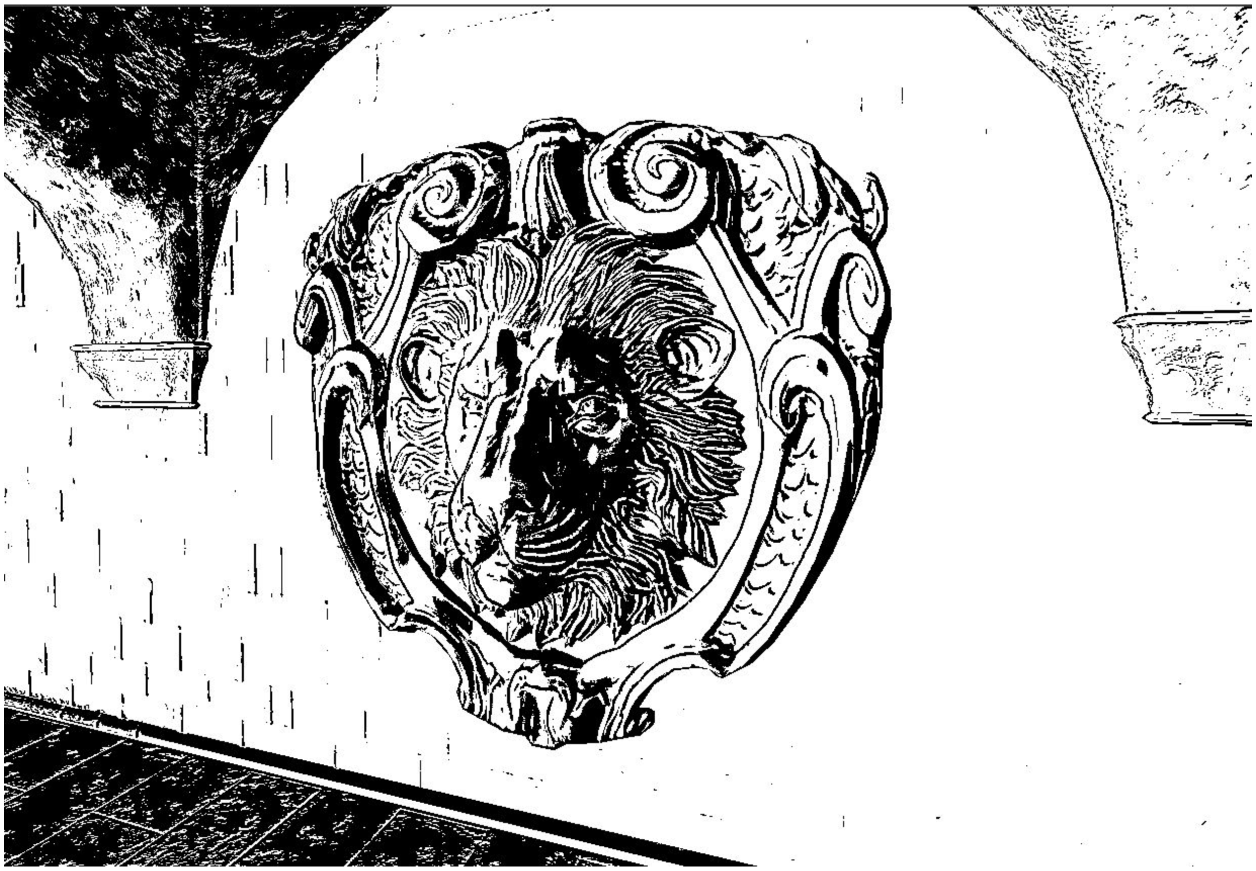
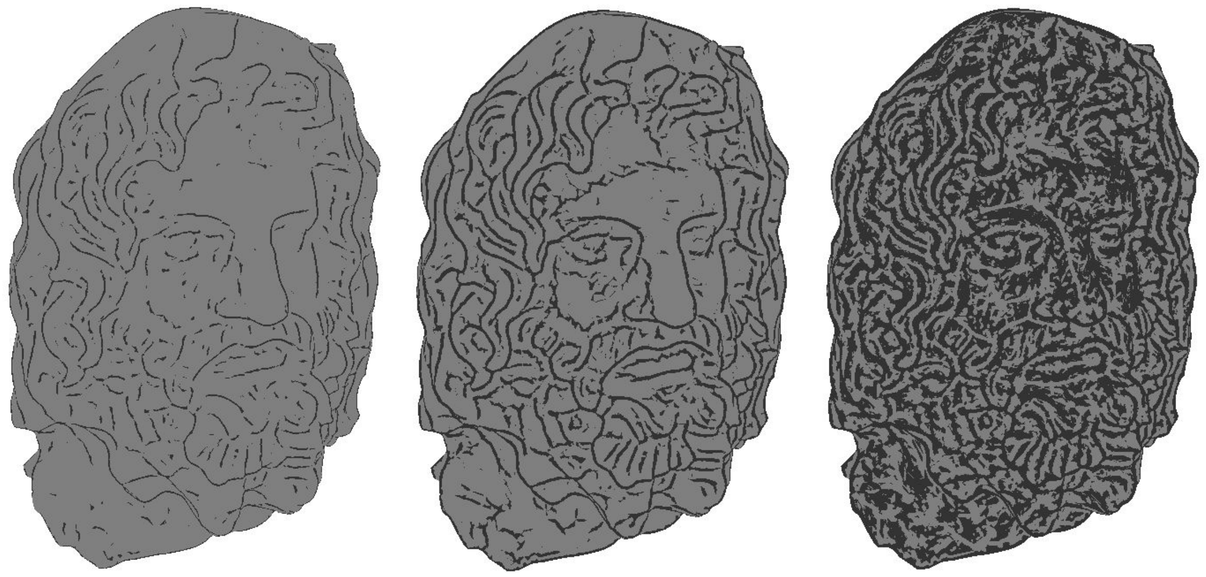


Image taken from my framework featuring an interesting wet-like effect of the Serapis head. On the left you can see the setting used to achieve such an effect.


Help
Scenes
Window
G-Buffer
displaying: lines
main thread frame average: 0.0 [ms]
render thread frame average: 48.2 [ms]

This image has been taken from my framework. It features a corner of the sponza using the Frank Miller's style preset. Sponza does use proper normal maps.


This image has been taken from my framework. It features the lion head of the sponza in Frank Miller's style preset. Sponza does use proper normal maps.



Three images taken from my framework. Using only the suggestive contour, an interesting effect happens when setting the minimum dot product difference to 0 and slowly raising the smaller than minimum allowed proportion. Try it out! The three images represent that transition going from left to right.

11. References

- The papers the project is based upon:
  - DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. **Suggestive contours for conveying shape**. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22, 3, 848– 855.*
  - DECARLO, D., RUSINKIEWICZ, S. 2007. **Highlight lines for conveying shape**. *NPAR*

- The API used in my framework:
  - OpenGL
- The libraries used in my framework:
  - Glad
  - OpenGL Mathematics (GLM)
  - stb_image
  - Open Asset Import Library (Assimp)
  - Json-cpp
  - Dear ImGui