Soft Peaks

The effects of smoothing on ray reflections

Lucas Frans Willem Furer

Supervisor(s): Dr. Ricardo Marroquim, Yang Chen

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2023

Name of the student: Lucas Frans Willem Furer
Final project course: CSE3000 Research Project
Thesis committee: Dr. Ricardo Marroquim, Yang Chen, Daniël Pelsmaeker

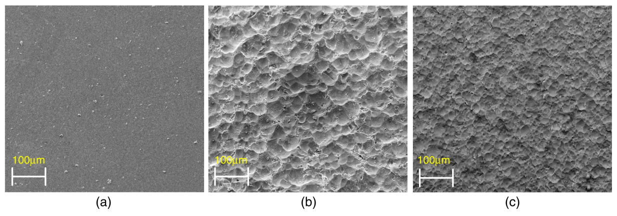
Abstract

The micro detail on surfaces can have a profound effect on how rays bounce of it. In ray tracing, this micro-surface is normally approximated with statistical models. We wish to figure out what the effect of normal interpolation and Phong tessellation is on how rays reflect on the surface. This has been Done by creating a ray tracers from scratch, that uses height fields, with normal interpolation and Phong tessellation. This ray tracer has then been used to render 3 different micro-surfaces, which were compared in order to understand the difference. The conclusion of this research was that both smoothing techniques had little to no effect on the reflection of rays on micro-surfaces with a high triangle density. However on micro-surfaces with a low triangle density, there was a much more pronounced difference with the smoothing techniques. A second conclusion was made which was that rougher surfaces have a more spread out distribution of rays than smoother surfaces.

1 Introduction

Not all surfaces that appear smooth on the macro level, are smooth on the micro level. A plethora of examples can be found when searching for SEM (scanning electron microscope) images. An example of this can be found in Figure 1 [1]. The roughness in materials can have a profound effect on their interaction with light. This can influence whether a surface has mirror like properties or not. In the field of ray tracing this micro-surface is often approximated using statistical models in order to reduce rendering time.

Figure 1: SEM image of an aluminum surfaces

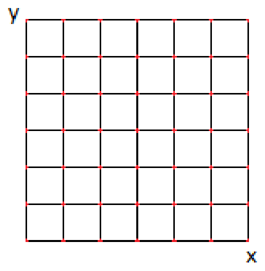


[1]

Ray tracing is a technique used to generate images of 3D models with realistic light interactions. Ray tracing works by creating a ray for every pixel on the screen, which will then hit and bounce on the surface of an objects and calculate what color that pixel should be. This has many uses, from games to animations and much more [2].

However, a big downside to ray tracing is that it often has a severe cost in terms of performance. Due to these performance costs, there has been a lot of research on trying to come up with techniques to improve the speed of ray tracing algorithms. For example, under some assumptions of the model, ray-tracing can be optimized. This is the case of height fields. While ordinary ray tracers use triangles to create any arbitrary shape, height fields approximates geometry by replacing triangles with a 2D array that determines what the height should be at that point [3], see Figure 2 where each red dot represents a value in the height field. This allows for a significant increase in performance. Nevertheless, height fields are still a piecewise linear representation of the surface.

Figure 2: grid structure where each red dot is a value



There exist techniques to render flat triangles in a smoothed way. This means that a model that exists of visibly flat triangles, can be rendered such that it looks smooth. The smoothing techniques that we shall consider in this paper are normal interpolation [4] and Phong tessellation [2]. These techniques apply a different kind of smoothing that can be combined to compliments each others short comings.

During ray tracing, when a ray is shot at a surface from the camera, it can reflect of the surface and continue to bounce around until it escapes (until it does not hit anything). The kind of reflection we shall use is called a polished reflection, which is the same kind of reflection as a mirror. This is normally done to collect information about the surrounding of the ray-surface intersection. In our case, we are only interested in the direction of the ray after it is done bouncing around.

The purpose of this research paper is to figure out what the effect of the previously mentioned smoothing techniques are on the reflection of rays on micro geometry. Furthermore, this must be done in a ray tracer that uses height fields as an input. Due to the specific focus of the current subject, there has not been a lot of research exploring the possibilities. To elaborate, there is sufficient research on height field ray tracers [3], normal interpolation [4] and Phong tessellation [2]. Yet it has proven difficult to find existing research which combines these concepts in order to study ray reflection on micro-geometry. One reason this subject is worth pursuing is that it might lead to better approximations of statistical models.

All this leads us to the following research question: what is the effect on the reflection of rays on micro geometry when applying smoothing techniques on a height field? This research question has been divided into the following sub-questions: What are the effects of using normal interpolation, Phong tesselation and a combination of those two techniques on the reflection of rays on a micro surface.

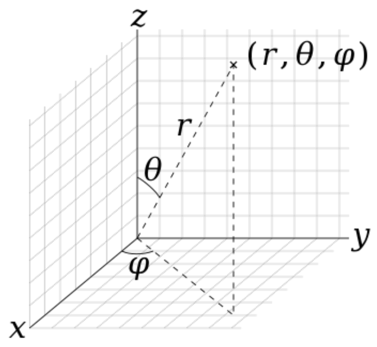
2 Methodology

In order to answer the research question, we must show the results of all the smoothing techniques so that we may compare them. A rendered image is usually the result of ray tracing, in our case, the results will be the camera rays that are reflected of a micro-surface. This can be done by taking a single micro-surface, applying the different techniques and comparing the results. This will then be repeated for varying micro-surfaces. The micro-surfaces themselves were obtained from the following paper [5].

There is one thing to note about the experimental setup, which is that an orthographic camera was used. Meaning the rays shot from the camera all have the same direction but different starting positions. This is the opposite of a perspective camera where the starting point is the same but the directions are different. The orthographic camera is used because the micro-surface can be approximated by a single point and thus the direction of all the incoming rays will be the same. For the same reason, the position of the reflected rays does not matter, since all the points on the micro-surface are approximated by a single point, only the direction matters.

In order to compare the the reflected rays, we must be able to visualize these rays. As has been stated earlier, only the direction of the reflected ray is of importance. Because of this, the the normal x, y, z representation of a ray rather inconvenient for our purposes. A much more convenient way to represent rays is with polar form. This form consist of two angles that determine a direction and a length along that direction. See Figure 3 for further clarification [6]. Since we have no interest in the length of the vector but only its direction, we can discard the length part of the polar form.

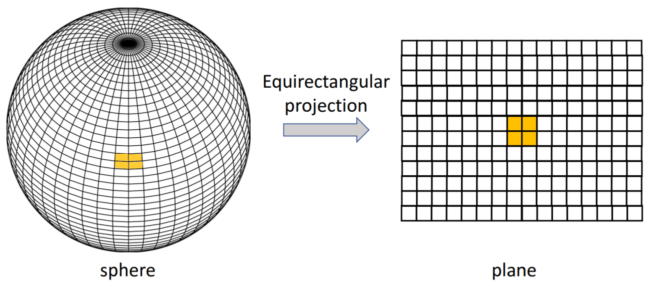
Figure 3: polar coordinates visualised



[6]

Once all the angles of the reflected rays have been calculated, we can plot them on a graph with the x- and y-axis being the two angles of the direction. Plotting the angles in a naive way (by grouping ray together that have a certain range in their angles) will result in a similarly deformed representation of a globe as common maps of the earth, see figure 4 [7]. This deformation can however be tolerated since the deformation will be the same for all of the graphs. As a result, the y-axis represents how much a ray is pointing upward. thus rays that are pointing upwards will be on the top half of the graph and those that point downward will be at the bottom half.

Figure 4: deformation of mapping vectors to a plane

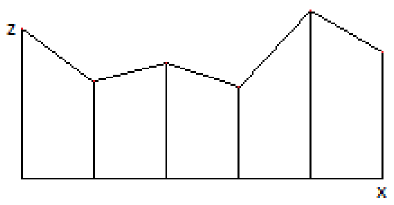


[7]

3 Heightfield

A height field is a data set composed of a 2-D array of numbers where the index corresponds to the location and the value to the height at said location. This special data set allows ray tracers to very quickly render a scene by taking advantage of its properties. See Figure 5 for a two dimensional version of a one dimensional height field for clarification.

Figure 5: 1 dimensional heightfield for a 2 dimensional world



The height field which is used for 3 dimensional renders is 2 dimensional and thus forms a grid when looked at from above, see figure 2. Each square, consisting of four points on the height field, that makes up this grid is called a cell.

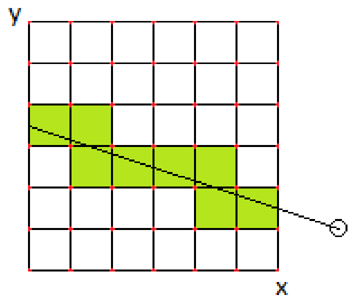
The implemented algorithm that traverses this height field was inspired by ,but not the same as, the grid tracing algorithm of Musgrave ([3]). How the algorithm traverses the height field is by going through each cell in order that is hit by a ray. This works by going through the following steps:

1. find intersection of camera ray with the bounding box of the height field.
2. determine in which cell the rays enters (keep track of the height of the ray)
3. calculate the coordinates of where the ray exits the cell (also keep track of the height at this point)
4. if the height of any of the four corners of the cell are between the heights of the entry and exit point, then perform an intersection test
5. if the intersection tests does not hit anything, then repeat from step 2 with the next cell that the camera ray hits. If no more cells can be hit then move on to the next ray

3.1 Grid traversal

At first, it might not be apparent how the traversal of this grid improves rendering time. This will however become clearer when we show how many cells in the grid have to be tested. In Figure 6 we show every cell that is hit and subsequently checked for intersection for a given ray. In normal ray tracers, a ray will simply check every triangle (or in our case every cell) to see if there is an intersection. This means that every cell that is not hit by a ray is another intersection test that will not have to be performed and thus safe time on rendering.

Figure 6: grid traversal with green highlight for hit cells



3.2 Cell-ray intersection

Calculating the intersections of the ray with the cells is an important part of this algorithm. It works by checking for intersections with the four sides of the cell. This will always result in two intersections. Unless we hit a corner resulting in one intersection or if we hit a side and are parallel with it resulting in infinitely many intersections. These two points are needed for step 4 of the algorithm. An intersection with one side, in this example we calculate the intersection with one of the vertical sides of the cell, is done by solving the following formula:

```
Dx = x component of the direction of camera
Ox = x component of the origin of the camera
R = length of the ray
X = the x value of one of the sides

R*Dx + Ox = X
which becomes
R = (X-Ox)/Dx
```

With the calculated R we can determine the coordinates of the hit point. However there is still an additional check that must be done to determine if the hit point is actually on the side and not on a line parallel to it.

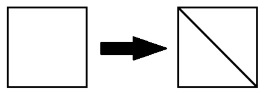
```
Ystart, Yend = the start/end value of the side
hitpoint = R*D + O

Ystart <= Yhitpoint <= Yend
```

Once this is repeated for all four sides. We can return the coordinates of the hit points. Do note that the x and y components in the formula have to be switched if we check for the horizontal sides of the cell.

3.3 Triangle-ray intersection

In step 4 of the algorithm we have to perform an intersection test. This intersection is not done with the square cell. Rather, the cell is split diagonally into two triangles. The result of this split can be seen in Figure 7.

Figure 7: transformation of a cell into two triangles



Once we have our two triangles we can perform two separate intersection tests [8]. If there is an intersection with both triangles, then we use the intersection with the smallest R (thus it will be in front of the other hit point and cannot be seen).

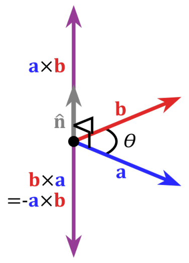
4 Normal interpolation

Normal interpolation is a surface shading techniques used to make a model (made out of flat triangles) appear smooth. This works by adjusting normal vectors in such a way that the surface appears smooth. This smoothing is however a sort of illusion since the actual geometry of the model remains the same. Because of this there are also some artifacts.

4.1 Normal vectors

Normal vectors are assigned (or calculated) up directions of triangles. These vectors determine what way a triangle is facing. In order to calculate the normal of a triangle, you can use the cross product of two different vectors that lay in the surface of said triangle. This will result in a vector that is perpendicular to the two vectors and thus create a correct normal vector. See Figure 8 [9]. There is one thing to note which is that the order of a cross product matters and if done wrong, can result in a vector that points in the wrong direction. Therefore, when calculating a normal vector, the correct order of vectors must be used in order to make sure the normal points upwards.

Figure 8: cross product between vector a and b



[9]

4.2 Uses

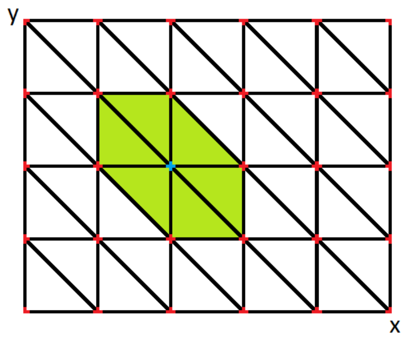
The purpose of normal vectors is to determine how light (or any ray in general) will reflect off its assigned triangle. This is useful to alter the way a material reflects light. In our case we shall be using the normal vectors to make our model look smooth. In other cases normal vectors can also be used to imitate details.

4.3 Assigning normal vectors

The normal interpolation algorithm works by assigning a normal vector to each point in the height field. This means that each triangle shall have 3 normal vectors on each of its corners. The algorithm to calculate these normals for each point works as follows:

1. obtain all the triangles that use the current point, see Figure 9.
2. calculate the normal vectors of these triangles using the cross product method.
3. average these normal vectors and assign them to the current point

Figure 9: example of which triangles are slected when obtaining normal vectors. The blue dot represents the selected point in the height field



4.4 Interpolation

Once a ray intersects with a triangle, we can calculate where exactly on the triangle it hit. This hit point can also be expressed in the following form where a, b and c are the three vector positions of the triangle.

```
hitpoint = a + beta*(b-a) + gamma*(c-a)
```

The meaning of the beta variable is how much of the vector from a to b is needed to get to the hit point. The same goes for the gamma variable except then for the vector from a to c. These two together can get to any point on the triangle. The reason for using these variables is that it tells us how close the hit point is to each corner. So a higher value of beta will result in the hit point being closer the the b vector of the triangle. We can also obtain an alpha variable (that determines how close we are to the a vector) with the following formula:

```
alpha = 1-beta-gamma
```

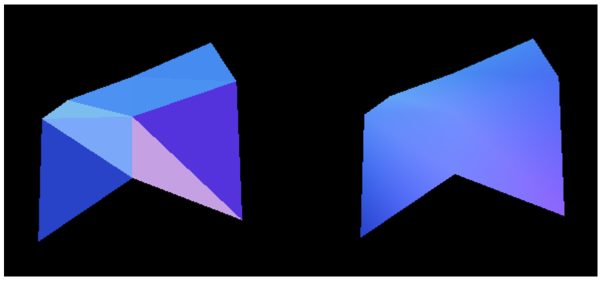
The alpha, beta and gamma variable together from the barycentric coordinates of the hit point on the triangle.

These barycentric coordinates are then used to obtain a new normal vector for that specific point by interpolating the normal vectors of the triangles corners.

```
NA = normal vector of point a
NB = normal vector of point b
NC = normal vector of point c

normalNew = alpha*NA + beta*NB + gamma*NC
```

This new normal vector will then be used for the ray reflection and the shading thus creating a smooth transition from the normal of one corner to the others. In order to visualise this, Figure 10 shows an example where the normal on each hit is used as a color.

Figure 10: small surface with randomized heights. Left without smoothing and right with smoothing. Normal vectors have been used as coloring



5 Phong tessellation

Phong tessellation is a smoothing techniques that divides triangles into smaller pieces and then offsets those new points to form a smoother surface. It was created by Boubekeur and Alexa [2]. There are 2 main parts of this algorithm. The tessellation of the triangles (dividing them) and the offsetting of the newly generated points.

5.1 Tessellation

Tessellating a triangle means to divide the triangle into smaller parts. This is done to allow more freedom on how the triangle looks. A triangle can be divided more or less depending on how much smoothing needs to be applied. In our case this is done by first setting a number of new points along the sides of the triangle that are equidistant, see Figure 11.

These new points are then used to divide the triangle into new smaller pieces as can be seen in Figure 12.

The more points we initially put on the sides of the triangle, the more we can smooth out the surface.

Figure 11: Points on triangle are equidistant

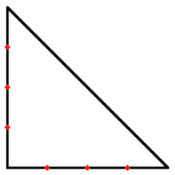
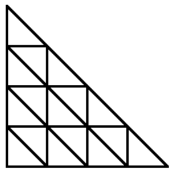


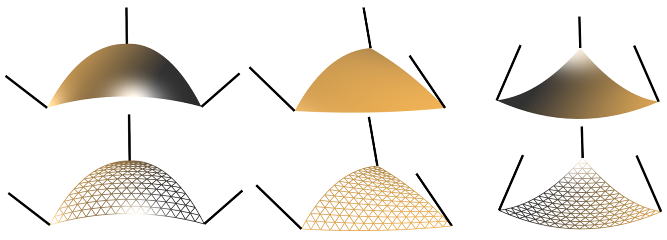
Figure 12: The result of the tessellation



5.2 Change point location

Just tessellating a triangle is not enough however. The only thing this has accomplished is that we now have more triangles that still make one big flat triangle. Each of the new generated points has to be put into a new position in order to create our smooth surface. For this we need 3 normal vectors in the corner of the original triangle in order to determine how we want the smoothing to look like. For example do we want a saddle, concave or convex shape, see Figure 13 [2].

Figure 13: result of Phong tessellation for differing normal vectors



[2]

How to obtain the normal vectors has already been explained in the normal interpolation section. The algorithm for offsetting each of the points works as follows with the original triangle consisting of the vectors Vi, Vj and Vk:

1. project the point orthogonally onto the plane defined by the point Vi and the normal vector at Vi
2. repeat step 1 for the point Vj and Vk
3. use the barycentric coordinates of the original point (U,W and V) to combine the 3 projected points

See Figure 14 for a visualization of the process [2].

See Figure 15 for a rendered comparison of Phong tessellation

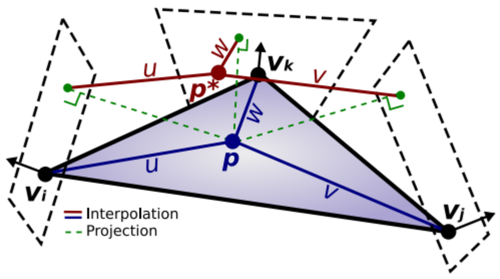
6 Responsible Research

The content of this paper is heavily focused on computer graphics. It also has had no need to involve people in its experiments at any point, thus limiting potential ethical issues. However, there are still some issues that occur even when dealing with something as exact as computer graphics. The things that are relevant in our case are proper citing and reproducibility of results. The correctly citing of sources is simply something that requires close attention. The reproducibility of results has been ensured in two ways:

The codebase [10] has been uploaded to a repository and can be freely downloaded.

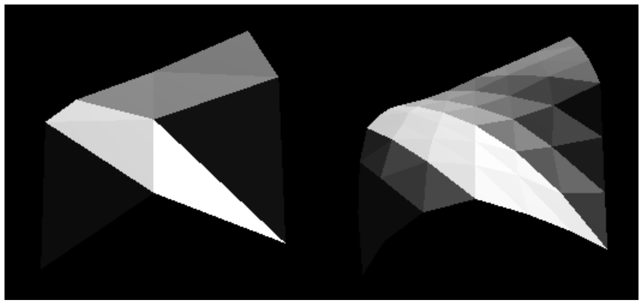
Furthermore, the methodology has clearly explained how the experiment will run. More exact details have also been provided in the results section allowing for the experiments to be replicated.

Figure 14: representation of the Phong tessellation



[2]

Figure 15: Left: a render without Phong tessellation. right: a render with Phong tessellation



7 Results

The results have been grouped together based on what surface was used. Furthermore, each surface has been rendered four times with the differing smoothing techniques. Each render has created a graph of the reflected (blue) camera rays and and image of the surface. Each of the graphs has one red dot in it, which represents a ray in the opposite direction of the camera ray. This helps us identify how the initial camera ray is transformed into the blue reflected rays. The camera points down by 45 degrees in each render. The surfaces that have been used are the al_1bar[5], the al_65bar[5] and one randomly generated surface. The al_1bar and the al_65bar differ in their roughness, where the latter is rougher than the former.

Figure 16: **smoothing:** no smoothing, **surface:** al_1bar[5], **size:** 480 by 640

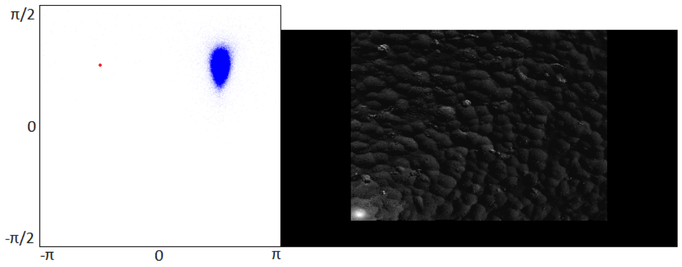


Figure 17: **smoothing:** normal interpolation, **surface:** al_1bar[5], **size:** 480 by 640

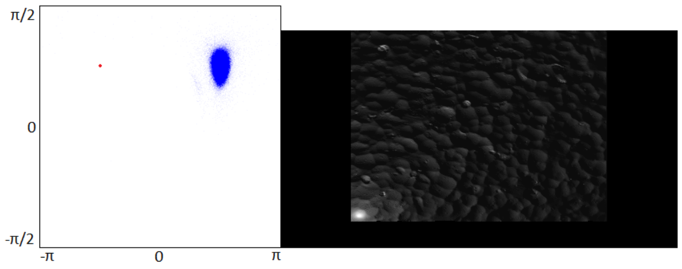


Figure 18: **smoothing:** Phong tessellation, **surface:** al_1bar[5], **size:** 480 by 640

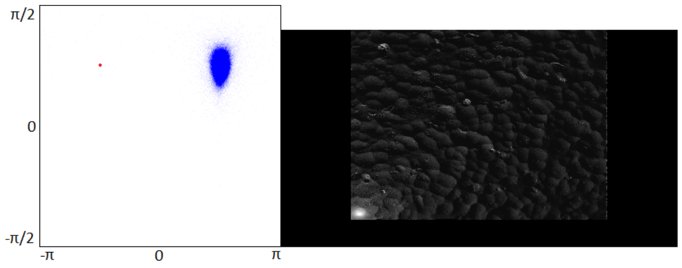


Figure 19: **smoothing:** normal interpolation and Phong tessellation, **surface:** al_1bar[5], **size:** 480 by 640

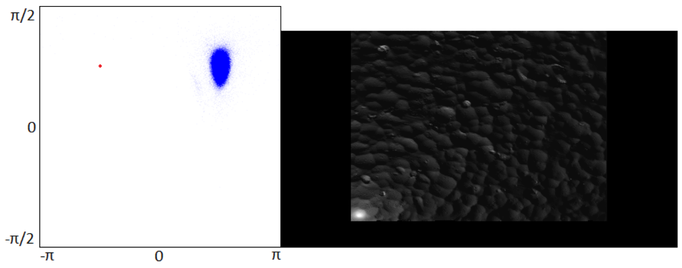


Figure 20: **smoothing:** no smoothing, **surface:** al_65bar[5], **size:** 480 by 640

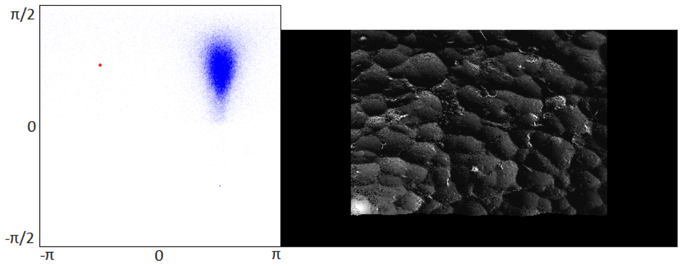


Figure 21: **smoothing:** normal interpolation, **surface:** al_65bar[5], **size:** 480 by 640

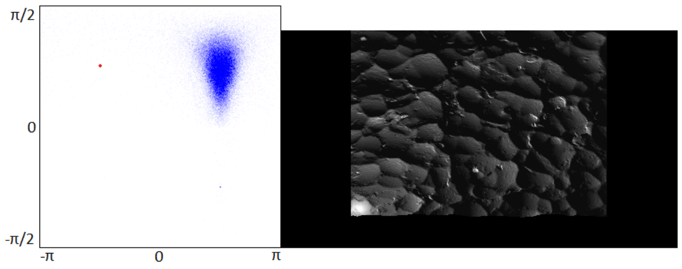


Figure 22: **smoothing:** Phong tessellation, **surface:** al_65bar[5], **size:** 480 by 640

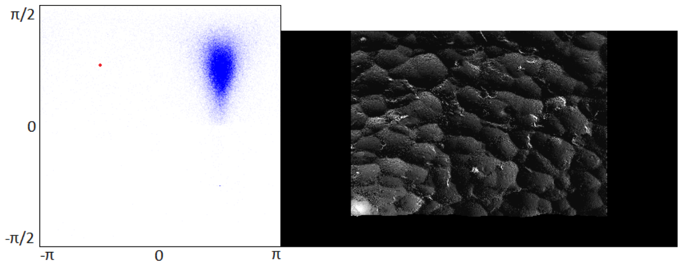


Figure 23: **smoothing:** normal interpolation and Phong tessellation, **surface:** al_65bar[5], **size:** 480 by 640

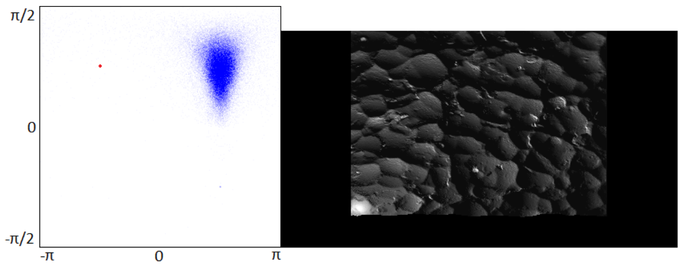


Figure 24: **smoothing:** no smoothing, **surface:** random, **size:** 30 by 30

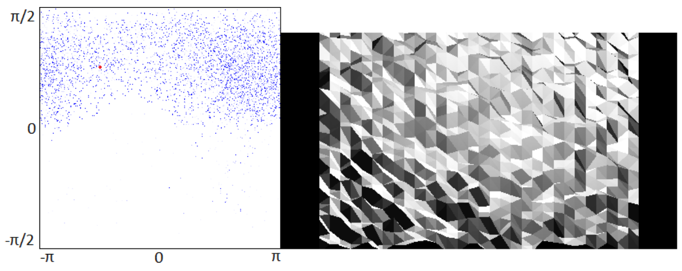


Figure 25: **smoothing:** normal interpolation, **surface:** random, **size:** 30 by 30

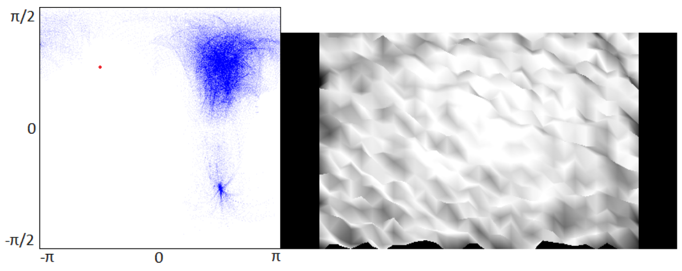


Figure 26: **smoothing:** Phong tessellation, **surface:** random, **size:** 30 by 30

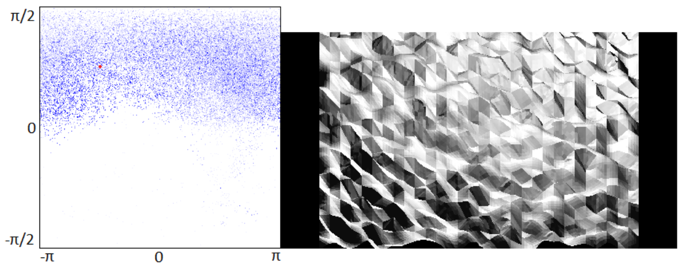
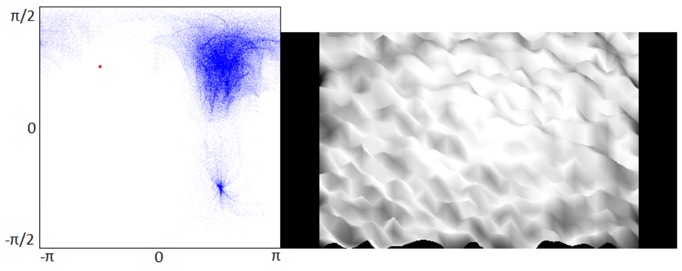


Figure 27: **smoothing:** normal interpolation and Phong tessellation, **surface:** random, **size:** 30 by 30



8 Conclusions

What is the effect of smoothing techniques on how rays reflect of a micro-surface. That is the question that we set out to answer. The generated results suggest two concepts of note.

When we look at the larger surfaces, the surfaces with size 480 by 640 (see Figure 16-23), we see that there is little to no difference in their reflection graphs when the smoothing is changed. We also see almost no difference between the rendered images of the same surface. The conclusion we draw from this is that smoothing techniques do not have a large impact on the reflection of light on surfaces with a high triangle density. The fact that there are more triangles in the surfaces than there are rays being shot means that it is highly unlikely that multiple rays bounce on the same triangle. Therefore, adding more triangles with Phong tessellation or by altering the normals with normal interpolation will not result in many more unique ray directions. This does change however when we look at the smaller surface of 30 by 30 (see Figure 24-27). Here we can see a significant difference in both the rendered image and the ray reflection graph. When no smoothing (see Figure 24) is applied we can see that the ray reflect into lots of dense directions. This makes sense when we know that multiple rays will likely hit the same triangle resulting in lots of small groups. When Phong tessellation (see Figure 26) is applied on this small surface, we see that these groups are split into more groups. This is consistent with our previous explanation because there are now more triangles. What we see with the normal interpolation (see Figure 25) is that the rays tend to group up in a direction away from the camera (notice how the concentration of points is horizontally π away from the red dot). When Phong tessellation and normal interpolation (see Figure 27) have been combined they look similar to just normal interpolation. One oddity that Can be noticed on Figure 25 and 27 is that there is a second (smaller) group of points below the main group. This means that a significant amount of rays reflect downward. This should not be possible unless the rays somehow go through the surface. The most likely explanation of this is that there is a floating point precision issue that causes some rays to end up below the surface when being reflected. In short, smoothing techniques matter more on surfaces with a lower triangle density.

Comparing the graphs of the al_65bar (see Figure 20-23) and the al_1bar (see Figure 16-19) surface. We can see that the reflected rays of the al_65bar are more spread out and have a more gradual boundary with where the rays do not go. This stays consistent throughout all the smoothing techniques. What we can conclude is that a rougher surface will result in a more spread out distribution of reflected rays.

8.1 Future work

There are still many more areas on this particular subject to be studied that have not been covered in this paper. Studying how the reflected ray distribution changes as the camera angle changes has not been covered and could yield interesting results. Improvements to the code can also be made. These improvements include increasing the speed of the renders and fixing the bug where rays penetrate the surface and go below it. The increased speed is useful in order to render larger surfaces.

References

[1] C. Zhai, D. Hanaor, G. Proust, and Y. Gan, "Stress-dependent electrical contact resistance at fractal rough surfaces," *American Society of Civil Engineers*, 2015. DOI: http://dx.doi.org/10.1061/(ASCE)EM.1943-7889.0000967.

[2] T. Boubekeur and M. Alexa, "Phong tessellation," *ACM Trans. Graph.*, vol. 27, 2008.

[3] F. K. Musgrave, "Grid tracing: Fast ray tracing for height fields," *Yale University Dept. of Computer Science Research.*, 1988.

[4] C. W. A. M. Overveld and B. Wyvill, "Phong normal interpolation revisited," *ACM Transactions on Graphics (TOG)*, vol. 16, 1997. DOI: 10.1145/263834.263849.

[5] O. Clausen, Y. Chen, A. Fuhrmann, and R. Marroquim, "Investigation and simulation of diffraction on rough surfaces," *Computer Graphics Forum*, vol. 42, no. 1, pp. 245–260, 2023. DOI: https://doi.org/10.1111/cgf.14717. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14717. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14717.

[6] Wikipedia, *Spherical coordinate system — Wikipedia, the free encyclopedia*, 2023. [Online]. Available: %5Curl%7Bhttps://en.wikipedia.org/wiki/Spherical_coordinate_system%7D.

[7] X. Jiang, Y.-H. Chiang, Y. Zhao, and Y. Ji, "Plato: Learning-based adaptive streaming of 360-degree videos," Oct. 2018, pp. 393–400. DOI: 10.1109/LCN.2018.8638092.

[8] S. Marschner and P. Shirley, *Fundamentals of Computer Graphics, Fourth Edition*, 4th. USA: A. K. Peters, Ltd., 2016, ISBN: 1482229390.

[9] Wikipedia contributors, *Cross product — Wikipedia, the free encyclopedia*, [Online; accessed 23-June-2023], 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Cross_product&oldid=1155788663.

[10] L. F. W. Furer. [Online]. Available: https://github.com/LucasFurer/Research_project_ray_tracer.git.