Pen and Inking Technique for Rendering Hermite Radial Basis Functions

ARYA BANAEIZADEH, University of Calgary, Canada

In this report, we are presenting an inking technique for rendering implicit surfaces. We will try to answer the question of how rendering implicit surfaces can be done without making any geometric representation of them. We rely on existing research done on this topic as well as artistic stippling and inking techniques. Our main contribution is the computational optimizations that are done to existing methods to render the surfaces faster and more efficiently.

Key Words: Computer Graphics, Non-photorealistic Rendering, Implicit Surfaces

1 INTRODUCTION

Non-photorealistic rendering (NPR) refers to a wide range of techniques for representing computer-generated imagery that remove the constraint of photorealism. Instead, the goal is to explore suitable alternatives for representing images and to find creative and expressive techniques along the way [Isenberg et al. 2006].

The main focus of computer-generated imagery throughout most research is to aim for photorealism. However, it is not true to say that photorealistic rendering techniques are applicable to every niche. In some areas non-photorealistic rendering techniques may become more handy; the goal here is not to depict reality as is, but as what is "meaningful" in the image and how to communicate images adequately to accentuate important details [Gooch and Gooch 2001]. The techniques used in non-photorealistic rendering rely more on the artistic aspect of depicting images as opposed to the physical properties of light, materials, etc. Since artistic endeavour shares some commonality with the main motivation of NPR which was stated earlier [Hertzmann 2010].

For communicating a proper image, the subjected NPR technique should be able to provide important perceptual cues. These cues can include certain contours or object outlines, and shading to show the inner form, texture and lighting condition of the object. One of the techniques that can be used here is the **pen and inking** technique. Our goal in this report is to explore this technique and find alleyways as to how to improve the performance the some of the previous research. We are going to mainly focus on the work by Brazil et al. [2011]. The technique is suitable for implicit surfaces.

Implicit surfaces represent a mathematically accurate depiction of a certain model. A significant advantage to this is that many nice-to-have mathematical properties from the model will emerge that can be utilized in various ways. Just to name a few, with an implicit surface we can have continuous models, curvatures and principal curvatures, and local differential properties. These can be used in NPR techniques as well.

The main downside of this topic is that rendering implicit surfaces presents its own series of challenges; since we do not have geometric information about the surface itself it is difficult to determine where the points are. This is another rendering challenge that needs to be addressed. There is always the route of making a geometrical representation of the surfaces with approaches like marching cubes [Lorensen and Cline 1987] and dual-contouring [Ju et al. 2002] and then rendering them using the normal rasterization pipeline. This is something that we want to get away from. The goal here is to render the surfaces without making an explicit representation of them.

Author's address: Arya Banaeizadeh, arya.banaeizadeh@ucalgary.ca, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada, T2N 1N4.

Fig. 1. One of the results of the implemented system, showcasing the Stanford bunny model

2 BACKGROUND

The field of NPR is a well-established field, with decent literature on the topic. The topic became a force to be reckoned with in the late 1990s[Kumar et al. 2019]. Before that most image manipulation techniques occurred in image-space and not in object-space. The advent of object-space techniques for NPR opened many alleys to more research on the topic. Even today, many NPR techniques used today also borrow from the older research and do not differ in terms of the general idea, albeit they make improvements on the performance of the algorithms mainly by leaving some of the work to the GPU.

The non-photorealistic rendering of implicit surfaces has been a topic of interest in computer graphics. For this project, we are mainly considering The work by Bremer and Hughes [1998], in which they introduced a real-time technique for modelling silhouettes of implicit surfaces. The other paper we are interested in is the work of Brazil et al. [2011] as mentioned in section 1.

Hermite radial basis functions have been chosen as the implicit surfaces that are to be rendered. They were first presented by Macedo et al. [2011]. In this paper, they introduced the main approach for interpolating surfaces. This approach is also in the work of Brazil et al. [Brazil et al. 2010]. The implicit model that we use is

based on the latter paper, in which an altered version of the functions of Macedo et al. is used. A more detailed explanation of the surfaces is provided in section 6.1.

3 OVERVIEW

Three essential elements should be considered for expressing surfaces, how many drawing primitives should be drawn, where they should be placed and what is the approach to drawing them [Kim et al. 2009]. In the context of pen and inking and implicit surfaces, we have to figure out how to sample points on the implicit surface such that it will uniformly cover the whole model. We can then employ different techniques to depict the form and silhouette of the object. In the following, we will go through an overview of how the general idea of the approach taken in this project.

Initially, we will start with a group of control points, these control points can be generated or derived using multiple approaches as explained in section 4. Once we have a roughly uniform sample over the surface we can start refining the samples by instantiating a series of points out of each control point and projecting it back to the model. we can repeat this process with the newer generated points to create more points. This process can be repeated until we get a good uniform distribution of points over the surface.

Depending on how the subdivision of points into newer points is done, we can achieve different styles of surface depiction, for instance, one approach would be to sample in the principal curvature directions, which would result in line drawings on the surfaces. This sits opposite to aiming for sampling randomly in all directions on the surface, which results in a stippling effect. The sampling technique introduced here is the approach that is presented in the work of Brazil et al. [2011].

Once the sampling and the refinement process are done we can move on to rendering the surfaces. We can process the points using a standard graphics pipeline and employ the relevant techniques for classifying points as say silhouettes and regular points.

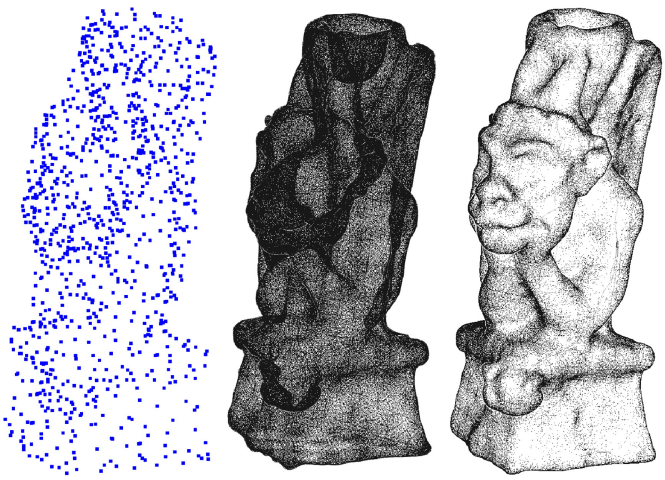


Fig. 2. The overall pipeline as explained in the original paper [Brazil et al. 2011]

4 SAMPLING AND REFINEMENT

In this section we will go over the details of placing seeds on the model and the refining process is done. Finally, we will mention the different sampling schemes that can be used for style variations.

4.1 Initial point placement

There are two main approaches that we will use for sampling points. The first approach relies on the original control points that have been used to interpolate the implicit surfaces. As HRBFs guarantee that the fitted surface will lie on all the control points that have been given to it we can reuse those points for the seed placement as well. This way, we will have the advantage of saving some computation and not worrying about projecting points on the surface. The downside is that if the initial sampling points are not uniformly distributed, the final render might not reflect the important cues of how the shape and form of the surface, since the subsequently generated sample points, are to be placed in close proximity to the original sample points.

If the control points are not uniformly distributed or for any other reason they cannot be used we can employ another approach; we take a regular grid such that it covers the entire model and then we generate a point in each cell with a random distance from the cell center. Finally, we will project the point on the model. The details of how the point projection on the surface is done are explained in section 6.2.

4.2 Refining Sample Points

Now that we have generated the control sample points it is time to generate rendering points from the series of points. The general approach is to generate a series of random points from each control point within a certain distance ρ and after projecting them on the surface we can repeat this process for the newly generated points by changing ρ to $\rho/3$. This process can be done as many times as is required to generate better images. We can generate these points using two approaches, stippling and direction sampling, both of which fit within the refining scheme.

To generate new points out of the already generated points, it is best to keep the new points as close as possible to the surface. This way the projection algorithm will perform better both in terms of robustness and also point-placement on the surface. For achieving a good approximation we can generate points on the incident tangent plane of the input points. We can use limit the domain of the points that are to be generated to a square patch over the input points where a generated point might have an offset of $x \in (-\rho, \rho)$ and $y \in (-\rho, \rho)$ on the local axis of the tangent plane. The size of ρ can be changed to fine-tune the results. For instance, if the initial samples are not roughly uniform ρ can be set to a higher value so that it is further from the current point samples. In our experience, if we set ρ to higher than need values the sampling will again become non-uniform since the points will get too far away from the original points.

In the following, we will explain the two different refining approaches and get into more detail about the sampling process.

4.2.1 Stippling. The first step in generating the stippling points is to calculate the tangent plane of the surface. For this we can create

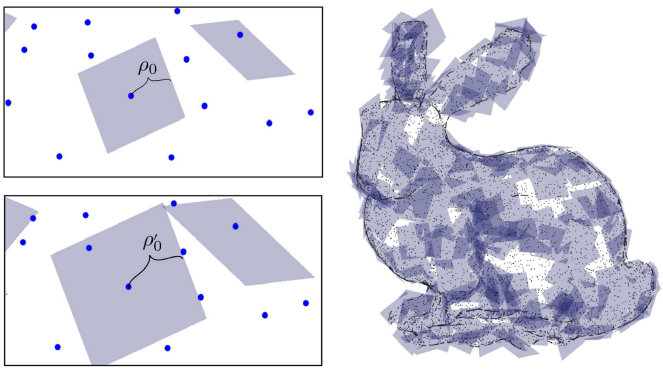


Fig. 3. Sampling points on the tangent plane of the control points as shown in the original paper [Brazil et al. 2011]

a local axis for each point; given that we have the point's normal we can rotate it with a rotation matrix R with an arbitrary axis and angle. From there we have $r' = Rn$, and then we can compute the two axes directions u and r on the plane with $u = n \times r'$ and $r = n \times u$. There are other ways to get calculate the frames. However, This approach has been carefully picked by Brazil et al. [2011] as they avoid patterns. we can move on to generate the random points with the help of u and r:

$$p_{i,j} = i.\rho r + j.\rho u,$$

Where $i, j = \pm 1 + m$ and m is a random number in $[-0.125, 0.125]$. This will result in 4 random points each on either side of the two axes of u and r. Repeating this process a few times will result in more stippling points and makes final renders more expressive.

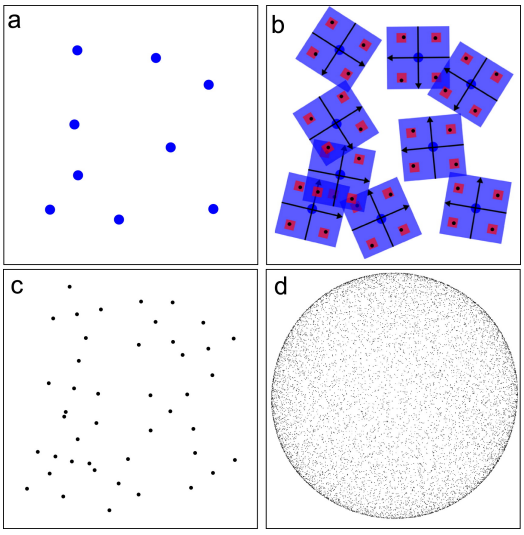


Fig. 4. Stippling sampling as shown in the original paper [Brazil et al. 2011]

4.2.2 Direction Sampling. The direction sampling follows the same idea as the stippling sampling, so we would need to generate the local axes for each point (*urn*). However, here instead of randomly

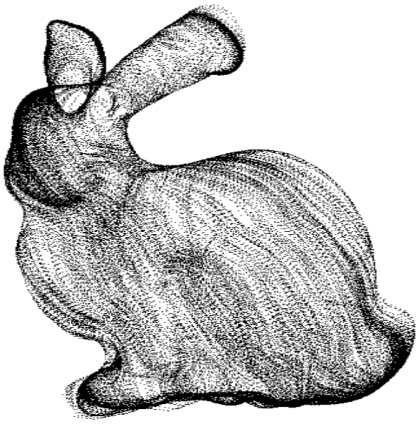


Fig. 5. Direction sampling following one of the axes' directions

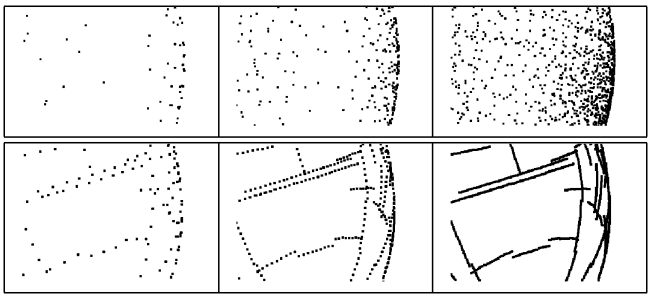


Fig. 6. Comparison of stippling vs direction sampling with different levels of refinement as shown in the original paper [Brazil et al. 2011]

sampling on the entire plane we chose either of the u and r and only generate points in that direction. To be more precise, the random points generated are computed from the following:

$$p_i = \pm \rho u,$$

or

$$p_i = \pm \rho r,$$

As we repeat this process we will start to see that the points will follow the line direction of either u or r.

Also further techniques can be employed to change the directions of u and r to give achieve different styles [Kindlmann et al. 2003], [Stark 2009].

After being done with the sampling process we can move on to the rendering stage.

5 RENDERING

Once we are done with the sampling process we can determine how the points should be rendered, i.e., what should be the colour value of each pixel in our final render. The idea is to transform the points to clip space so and show them on the screen; the process that is used in a typical rendering pipeline. The points will additionally be classified in either of the three categories, silhouettes, fronts and

(a) (b) (c) (d) (e)

Fig. 7. Figures showing refinement levels of stippling and their effect on the subsequently generated series of points

backs. with the distance vector of the point from the camera, v and the point normal ($\nabla f(p)$) n, we can determine which category the point belongs to:

- if $|n \cdot v| < 0.1$, the point is a silhouette.
- if $n \cdot v > 0.1$, the point is a front.
- if $n \cdot v < -0.1$, the point is a back point.

From here on out we can associate different colour values to the different point types.

Doing so however will not resolve the issue of occluded points on the object that are caused by self-occlusion. This problem also needs to be addressed. While the rendering of the points is being done, we can store their depth value in the alpha channel and drop the point if the alpha value of the pixel is already smaller than the current point.

6 IMPLICIT SURFACES

In this section, we will briefly go over the types of surfaces that we are depicting using NPR, as the properties of the implicit surface will also come in handy while dealing with the challenges of rendering these surfaces.

6.1 Surface Interpolation

Hermite radial basis function implicits(HRBFs) were first presented by Macedo et al. [2011]. This approach basically combines the theory of Hermite-Birkhoff interpolation with radial basis functions.

HRBFs use points and normals as the basis for generating surfaces. Here we go over the problem and describe how Hrbfs work in practice.

6.1.1 Theory. Given a set of n-dimensional pairwise distinct points $P = \{p_1, p_2, ..., p_m\}$ and their normals $N = \{n_1, n_2, ..., n_m\}$, HRBF will provide a function $f : \mathbb{R}^n \to \mathbb{R}$ which interpolates both P and N:

$$f(x) = \sum_{j=1}^{m} \{\alpha_j \psi(x - p_j) - \langle \beta_j, \nabla\psi(x - p_j)\rangle\} \tag{1}$$

where ψ is a radial basis function ($\psi : \mathbb{R}^n \to \mathbb{R}$), i.e., it should be a function of the distance between the two points, $\langle\cdot\rangle$ is the indicator of dot products between two vectors, and the gradient operator is shown as ∇.

Here, α_j is a scalar value and β_j is a vector ($\alpha_j \in \mathbb{R}, \beta_j \in \mathbb{R}^n$). Their values can be determined by setting the interpolation function equal to these values:

$$f(p_i) = 0 \;\; and \;\; \nabla f(p_i) = n_i, \quad with \; i \in \{1, 2, ..., m\} \tag{2}$$

In other words, the interpolation function and its first derivative will be set to zero and the normal vectors:

$$\sum_{j=1}^{m} \{\alpha_j \psi(p_i - p_j) - \langle \beta_j, \nabla\psi(p_i - p_j)\rangle\} = 0 \tag{3}$$

$$\sum_{j=1}^{m} \{\alpha_j \nabla\psi(p_i - p_j) - \beta_j \mathbf{H}\psi(p_i - p_j)\} = n_i \tag{4}$$

where $\mathbf{H}$ is the Hessian operator which is applied to ψ, and $i \in \{1, 2, ..., m\}$.

These sets of equations can be turned into a linear system and solved accordingly:

$$\mathbf{Ax} = \mathbf{B} \tag{5}$$

Where $\mathbf{A}$ is a $(n+1)m \;\; by \;\; (n+1)m$ matrix, while $\mathbf{B}$ and $\mathbf{x}$ are $(n+1)m \; by \; 1$ matrices.

If $A_{i,j}$ is the $n+1 \; by \; n+1$ block of the matrix starting from the ith row and jth column we will have:

$$A_{i,j} = \begin{bmatrix} \psi(p_i - p_j) & -\nabla\psi(p_i - p_j) \\ \nabla\psi(p_i - p_j)^T & -\mathbf{H}\psi(p_i - p_j) \end{bmatrix}$$

simlilarly we have for $\mathbf{B}$ and $\mathbf{x}$:

$$\mathbf{x_i} = \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$$

$$\mathbf{B_i} = \begin{bmatrix} 0 \\ n_i \end{bmatrix}$$

While this system will work most of the time, there is no guarantee that the system can be solved since the matrix $\mathbf{A}$ might not be positive definite (for more information please refer to [Wendland 2004]). Therefore, usually a linear element gets added to the interpolant function, changing equation 2 into the following:

$$f(x) = \sum_{j=1}^{m} \{\alpha_j \psi(x - p_j) - \langle \beta_j, \nabla\psi(x - p_j)\rangle\} + \langle a, x\rangle + b \tag{6}$$

Where a is an n-dimensional vector and b is a scalar value.

Equations 3 and 4 will be updated accordingly:

$$\sum_{j=1}^{m} \{\alpha_j \psi(p_i - p_j) - \langle \beta_j, \nabla\psi(p_i - p_j)\rangle\} + \langle a, p_i\rangle + b = 0 \tag{7}$$

$$\sum_{j=1}^{m} \{\alpha_j \nabla\psi(p_i - p_j) - \beta_j \mathbf{H}\psi(p_i - p_j)\} + a = n_i \tag{8}$$

In addition to above constraints we would also have to add to more constraints to αs and βs, more specifically the following should be satisfied:

$$\sum_{j=1}^{m} \alpha_j = 0 \tag{9}$$

$$\sum_{j=1}^{m} \{\alpha_j p_j + \beta_j\} = 0 \tag{10}$$

This way we are going to ensure that we get unique values for all the unknowns and thus interpolating a surface, and the matrix is invertible.

We have to reflect these changes in the matrix calculations; the following $n+1$ *by* $n+1$ rows and columns will be added to matrix $\mathbf{A}$:

$$A_{i,(n+1)m+1} = \begin{bmatrix} 1 & p_i^T \\ 0 & \mathbf{I_{n\times n}} \end{bmatrix}$$

$$A_{(n+1)m+1,j} = \begin{bmatrix} 1 & 0 \\ p_j & \mathbf{I_{n\times n}} \end{bmatrix}$$

where $\mathbf{I_{n\times n}}$ is an identity matrix.

Also, the following will be added to $\mathbf{B}$ and $\mathbf{x}$:

$$\mathbf{x_{(n+1)m+1}} = \begin{bmatrix} b \\ a \end{bmatrix}$$

$$\mathbf{B_{(n+1)m+1}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This interpolation was first introduced by Brazil et al. [2010].

6.1.2 Radial basis function. We use $||\mathbf{x}||^3$ as the radial basis function. This is also used in the work of [Brazil et al. 2010], but was first presented in [Duchon 1977]. Thus, the radial basis function and its gradient and hessian will be calculated as follows;

$$\psi(x) = ||x||^3, \quad \nabla\psi(x) = 3x||x||, \quad \mathbf{H}\psi(x) = \frac{3}{||x||}(||x||^2\mathbf{I_{3\times 3}} + xx^t)$$

6.2 Point Projection on Hermite radial basis functions

For projecting the points onto the surface we use a gradiant-based iterative approach to approximate the location on the surface. The rationale is explained in the work of Bertsekas [1997]. For projection of the point p on the surface we do the following:

$$p^0 = p, p^{k+1} = p^k - \delta^{i_k} \frac{f(p^k)}{||\nabla f(p^k)||^2} \nabla f(p^k) \tag{11}$$

where i_k is the smallest non-negative integer such that

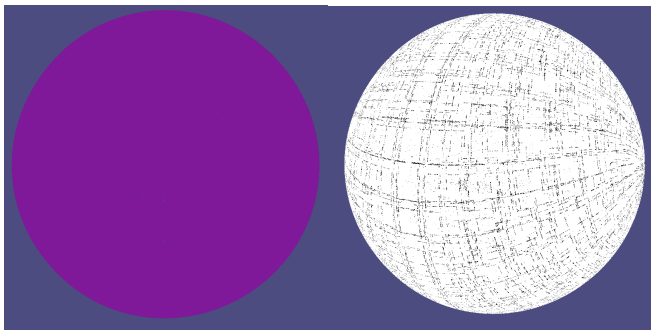
$$(f(p^{k+1}))^2 \leq (f(p^k))^2[1 - 2\phi\delta^{i_k}]$$

with $\delta \in (0,1)$ and $\phi \in (0, \frac{1}{2})$.

Doing this will reduce the step size as the iteration continues based on the *Armijo Rule* [Bertsekas 1997].

The iteration will stop under these conditions:

- $f(p) < \varepsilon_1$. p is returned.
- $\nabla f(p) < \varepsilon_2$. p is dropped.
- $i_k > 16$. p is dropped.



(a) (b)

Fig. 8. Figures showing stippling 8a and direction sampling 8b on a sphere model

6.3 Properties of Hermite radial basis functions

HRBF surface interpolation is a very powerful way of computing surfaces. We list some of the advantages that HRBFs have that influenced our decision to use them.

- **Implicit surface interpolation:** HRBFs give us a clear notion of the inside and outside of an object, which we can use to figure out whether a certain point lies on the surface or not. We can also determine the gradient of each point, which is useful for normal calculation and also projection.
- **Smoothness:** Hrbfs guarantee C^2 continuity on the control points of the surface and up to C^∞ continuity elsewhere. This continuity level depends on the continuity of the radial basis function.
- **Affine Invariance:** HRBFs ensure affine invariance, i.e., no matter how we rotate, scale or translate the control points the same surface is calculated. This is an important advantage as we would only have to compute the sampling and refinement points only once at the beginning of the program.

One major disadvantage of HRBFs is that the surfaces are globally defined, i.e., for generating the surfaces and evaluating the function value and the gradient we would need to take all the control points into consideration. This can be a heavy burden on the implementation side and make us lose performance. However, some processes can be parallelized.

7 RESULTS

In this section, we discuss some of the results that we achieved with the project.

7.1 A simple case

In our first tests, we implemented the approach fully on the CPU. The model test here was a simple sphere with 6 control points. In the samplings, 6 levels of subdivision are used both for direction sampling and stippling. Figure 8 demonstrates the sphere. For figure 8b the number of rendered points rounds up to 65.5k samples.

7.2 Stanford Bunny

Moving on to GPU coding, we mostly experimented with the Stanford bunny model. The main approach for rendering geometric models is to interpolate implicit surface first and then discard the geometric info of the model and only use the Hermite function. Below we will explain some of the results we got.

With 312 control points and 4 subdivision steps for both stippling and direction sampling, the preprocessing step took 12.4 seconds and we got up to 30 frames with a moving camera. The number of rendering points came to be 212.8k.

We changed the number of control points to 684 control points with the same model and did the process again. This time it took 22.5 seconds for the computing stage and we got up to 8 frames with a moving camera. The final number of points was 466K.

If we keep the samples at a decent number, around 250k, we would have a real-time interaction and can move the camera around the object and see the updates made to the rasterization process for rendering points.

7.3 Other models and variations

We also rendered two other models, a hand and a gourd. Figures 9 and 10 show the results of these. Finally, we have some style variations on the bunny model (figure 11).

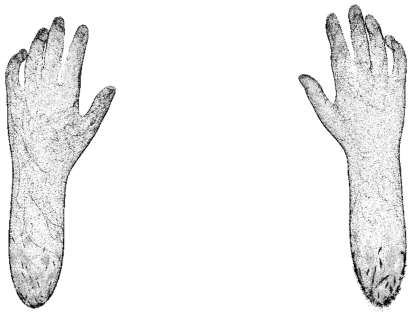


Fig. 9. A hand model with stippling

All of the models discussed and their corresponding points in the images are rasterized to a 1024×1024 texture.

8 IMPLEMENTATION

In this section, we will talk about the details of implementation and explain the attempt made to improve the performance of the work of Brazil et al. [2011]. This section would perhaps be the main point of contribution as the main approach does not differ that much from the paper.

We will start by explaining how the computations are done and how the workload is divided between CPU and GPU. The main technologies used for implementations are also briefly introduced.

8.1 CPU Bound Calcualtions

There are two main steps that can be time-intensive in the execution of the program before the rendering loop. Interpolating the HRBFs and the sampling and refinement process. In our project, the first step is done using the CPU. For the surface interpolation, we need to solve equation 5. This computation might turn into a bottleneck as the order of computation is $O(n^3)$.

If one intends to improve the performance of the interpolation, one can do so by attempting to parallelize some of the procedures on the CPU.

8.2 GPU Bound Calculations

The sampling and refinement process is done in GPU. The main motivation for moving this part into GPU is that as the number of rendering points increases, the number of computations will also increase exponentially. Thus, the more it can be parallelized, the more performance we can get as a result. Running the program on GPU is our best bet to do this optimization.

We mainly use to shaders refine the points and render them back to the user. In the following, we will explain how these shaders are implemented.

8.2.1 Point Shader. The point shader is responsible for a one-step refinement of the points. It inputs a series of points that at level zero are the initial control points on the surface. Then, it does the relevant calculations of generating the series of points needed for stippling or direction sampling. If we are using the stippling approach say, we would get four output points per one input point. For the projection step we require to have to know what $f(p)$ and $\nabla f(p)$ are, so these two functions should be implemented on the GPU. Our approach was to do these calculations in a compute shader and buffering the α_i and β_i coefficients on the GPU so that the function values can be evaluated.

For doing the refinement process multiple times, the shaders can be changed together, i.e., the outputs of the first compute shader can be given as the input of the second and so on. This way we can do all the point-related computations on GPU so long as we have enough memory.

Each invocation of the shader at any refinement step is responsible for calculating only a limited group of points. If we were to do

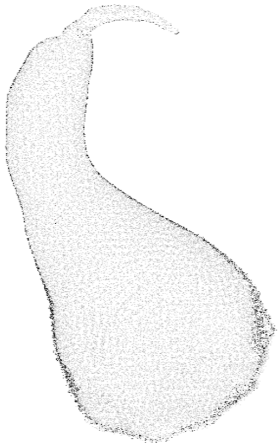
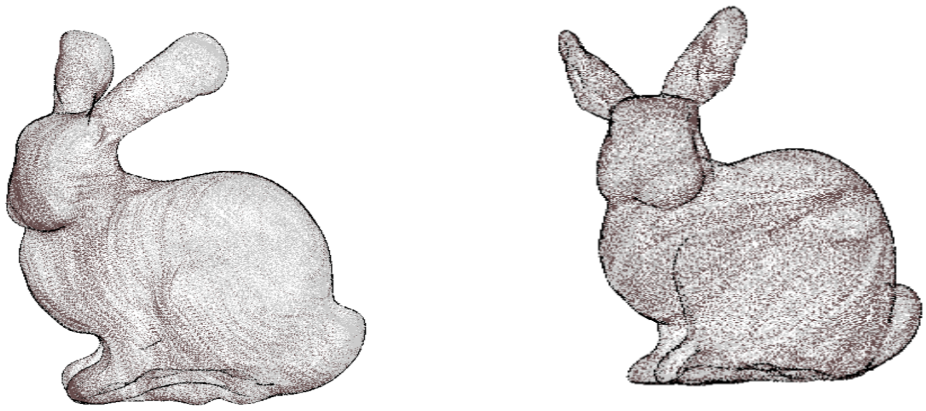


Fig. 10. A gourd model with stippling



(a) sampling in u direction

(b) sampling in r direction

Fig. 11. Stanford bunny model with stippling and direction sampling

this task on a CPU with no parallelization, the process would have been much more time-intensive.

8.2.2 Render Shader. After we are satisfied with the number of generated points we can invoke another shader that is used for drawing the points on the screen. This part is where the rasterizing process of our standard rendering pipeline will happen, as well as the rendering techniques discussed in section 5. This shader, despite the point shader, belongs in the rendering loop, as the model and view matrices will change if the camera direction changes. The same can be said for silhouette and front-facing and back-facing points.

8.3 Technologies

We have used **C++** as the main programming language for implementing the project. C++ is usually the go-to place for creating applications that involve heavy computer graphics concepts due to its high performance. **OpenGL** has been used for rendering and visualizing the graphical data. OpenGL is an API for rendering two- and three-dimensional vector graphics. OpenGL utilizes the graphics processing unit to achieve fast results.

glsl has been used as the shading language of the project. glsl is OpenGL's principal shading language and offers many powerful features for coding on GPU [Wiki 2021]. Most aspects of the project were implemented in glsl.

For doing matrix calculations, we have used **Eigen**. It is a C++ library for calculating matrices, vectors, and other related algorithms in the field of linear algebra. We will be using Eigen for calculating matrices. This is mainly used for the CPU-bound calculations of HRBFs.

9 CONCLUSION AND FUTURE WORK

In this paper, we discussed an approach for rendering implicit surfaces using stippling and point sampling techniques. We also talked about how this approach can be improved using the GPU.

There are many routes to take if one would like to improve this project. Just to name a few, further optimization on the operations, especially the CPU-bound ones can be done, meaningful changing direction of curvatures to get different styles and the implementation of hidden-line attenuation.

Also within this framework, we can take other interesting modifications to achieve ambient occlusion by other HRBF surfaces. The idea here would be to check the control points of surface A against the interpolation function of surface B to see if they share similar spatial coordinates. Since we can do most of these calculations on GPU there should be little to no concern for the robustness and efficiency.

REFERENCES

Dimitri P Bertsekas. 1997. Nonlinear programming. *Journal of the Operational Research Society* 48, 3 (1997), 334–334.

E Vital Brazil, Ives Macedo, M Costa Sousa, Luiz Henrique de Figueiredo, and Luiz Velho. 2010. Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. 1–8.

Emilio Vital Brazil, Ives Macedo, Mario Costa Sousa, Luiz Velho, and Luiz Henrique De Figueiredo. 2011. Shape and tone depiction for implicit surfaces. *Computers & Graphics* 35, 1 (2011), 43–53.

David Bremer and John F Hughes. 1998. Rapid approximate silhouette rendering of implicit surfaces. In *In Proc. Implicit Surfaces*. Citeseer.

Jean Duchon. 1977. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions of Several Variables*, Walter Schempp and Karl Zeller (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 85–100.

Bruce Gooch and Amy Gooch. 2001. *Non-photorealistic rendering*. AK Peters/CRC Press.

Aaron Hertzmann. 2010. Non-photorealistic rendering and the science of art. In *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*. 147–157.

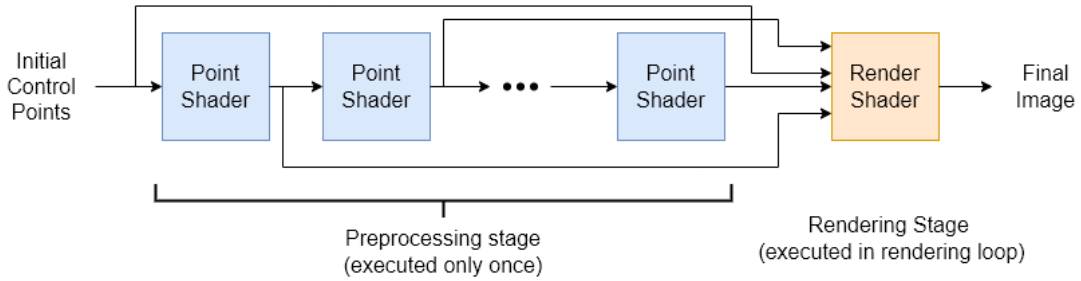


Fig. 12. The pipeline of the rendering process. As shown in the image the refinement process is done in the preprocessing stage using point shaders, and then the output of all the shaders is inputted to the render shader which in turn produces the final image

Tobias Isenberg, Petra Neumann, Sheelagh Carpendale, Mario Costa Sousa, and Joaquim A Jorge. 2006. Non-photorealistic rendering in context: an observational study. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. 115–126.

Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 339–346.

Sung Ye Kim, Ross Maciejewski, Tobias Isenberg, William M Andrews, Wei Chen, Mario Costa Sousa, and David S Ebert. 2009. Stippling by example. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*. 41–50.

Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller. 2003. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *IEEE Visualization, 2003. VIS 2003.* IEEE, 513–520.

MP Kumar, B Poornima, HS Nagendraswamy, and C Manjunath. 2019. A comprehensive survey on non-photorealistic rendering and benchmark developments for image abstraction and stylization. *Iran Journal of Computer Science* 2, 3 (2019), 131–165.

William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

Ives Macedo, Joao Paulo Gois, and Luiz Velho. 2011. Hermite radial basis functions implicits. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 27–42.

Michael M Stark. 2009. Efficient construction of perpendicular vectors without branching. *Journal of Graphics, GPU, and Game Tools* 14, 1 (2009), 55–62.

Holger Wendland. 2004. *Scattered Data Approximation*. Cambridge University Press. https://doi.org/10.1017/CBO9780511617539

OpenGL Wiki. 2021. OpenGL Shading Language — OpenGL Wiki,. http://www.khronos.org/opengl/wiki_opengl/index.php?title=OpenGL_Shading_Language&oldid=14750 [Online; accessed 20-April-2022].