Volume xx (200y), Number z, pp. 1–6

Transfer function design gallery

Seminar Assignment at the Advanced Computer Graphics 2020/21 Course

Žiga Babnik

Faculty of Computer and Information Science
University of Ljubljana
Večna Pot 113, SI-1000 Ljubljana
zb1996@student.uni-lj.si

Abstract

In the paper we present an automatic transfer function generation algorithm using the Intensity-Gradient Magnitude approach built into the Volumetric Path Tracing Framework. Gradient calculation is done using a separate script. From the input gradients and intensities we construct so-called bins. The bins contain spatial information regarding points with identical gradient and intensity magnitudes. We perform clustering over the information held by the bins using the KMeans algorithm. A transfer function is then constructed from the clustered data, which can be directly used in the Volumetric Path Tracing Framework. We present the time complexity of the implemented approach tested on different inputs. Additionally we also visually present the results given by the constructed transfer functions. Overall our approach produces transfer functions that in most cases correctly separate different materials within the input.

1. Introduction

A transfer function is an important part of volume rendering, determining how the input data is visualized. In order for a rendering to convey important information a transfer function needs to be designed properly. There are several ways of designing a transfer function, one of the more prominent is the *Intensity-Gradient Magnitude (IGM)* approach. The IGM approach relies on constructing an IGM histogram, which portraits the input data points in an intensity and gradient magnitude space. By clustering the IGM histogram a 2D transfer function can be obtained.

Volumetric Path Tracing Framework (VPT) [BM18] is an open-source visualization framework, designed for volume rendering. The framework operates as a web application, where the user can upload volumetric data and choose different renderers to visualize the data. It also supports the use of an Intensity-Gradient Magnitude (IGM) transfer function, which the user can define. Our goal is to extend the functionality of VPT to support automatic generation of transfer functions using clustering of the IGM histograms.

In Section 2 we present our approach, first in Section 2.1 we show how the gradients were calculated, in Section 2.2 we show how we construct the IGM histogram, in Section 2.3 we show how we clustered the obtained IGM histogram, in Section 2.4 we show how we constructed a transfer function and lastly in Section 2.5 we show how we construct new transfer functions from the original.

In Section 3 we present the results of our implementation, in Section 3.1 we present the time complexity of our implementation and in Section 3.2 we present some visualizations.

In Section 4 we summarize our approach and lastly, in Section 5 we present some weaknesses and possible upgrades of our approach.

2. Methods

In the following section we present how we extended the VPT framework to deal with automatic transfer function generation using clustering of the IGM histogram. To implement the transfer function design we took inspiration from the paper *A Clustering-Based Automatic Transfer Function Design for Volume Visualization* [ZYZ*16]. One main concern of our implementation is the time and space complexity since the framework is designed to run in a browser. As such we had to alter our implementation and avoid any potentially time or space consuming computation.

2.1. Gradient calculation

To calculate the gradients of given volumetric data, we implemented a *Python* script, named *gradient.py*. It takes as input the location of the *RAW* file containing the volumetric data and the x, y, z dimensions of the data.

The whole data is first read and mapped to float values. After which, we perform the gradient calculation using the *sobel* function implemented in the *scipy* library. The calculated gradients can be negative so we take absolute values of all calculated gradients. In the next step we normalize the gradients to the interval [0, 255], by first dividing all values by the maximum and than multiplying them by 255. The intensity and gradient magnitudes for all points are than stored sequentially into a flattened list. Finally we encode all the values in the list as binary values and write them to the output file.

Additionally we had to extend the *RAWReader* to support loading of volumetric data, where each point is described using two values.

2.2. IGM bins

Once an appropriate file has been generated as described previously in Section 2.1, the user can use the data in VPT.

Given some volumetric data represented as a 3D matrix $D^{x,y,z}$, VPT loads the data in flattened slices D_z^{x*y}, divided by the z-dimension. From the input data we construct bins, which hold information about all input data points with the same intensity and gradient magnitudes. For each bin we calculate the number of points assigned to it and the mean position, as well as the positional variance of all points assigned to it. Since holding all points in memory is not viable we calculate these values recursively. We calculate the mean position as shown in Equation 1, where μ_{i-1} is the previous mean value, i is the number of points assigned to the bins and p is the actual value we are adding to the mean. We calculate the variance as shown in Equation 2, where v_{i-1} is the previous variance.

$$\mu_i = \frac{\mu_{i-1}(i-1)+p}{i} \tag{1}$$

$$v_i = \frac{v_{i-1}(i-1)+(p-\mu_i)^2}{i} \tag{2}$$

2.3. Clustering

All calculations described in Section 2.2 take place on the client side of the application. Additionally we also implemented a new *Clustering* input field, which defines the number of clusters to be generated. Once the bin calculation is finished a *POST* request containing the bin data and number of clusters is made to the server.

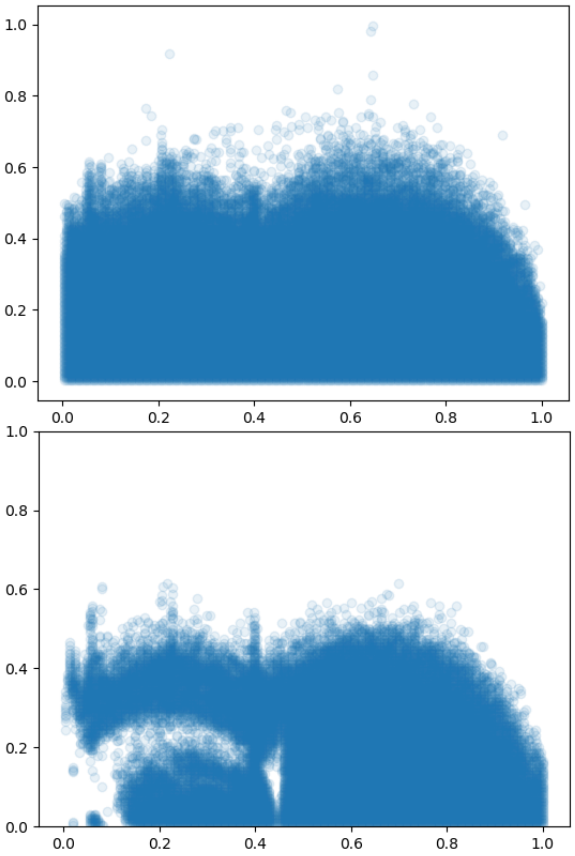

Figure 1: *IGM histogram without preprocessing (top) and the same IGM histogram with preprocessing (bottom)*

The clustering is then done on the server side using a Python script. Before we compute the actual clusters some additional preprocessing steps are taken. Firstly we clean the bin data by removing bins that contain less then two input data points. From the rest we calculate the mean variance values in all directions. If a bin has larger variance than the mean variance in any direction we remove it. Secondly we normalize the intensity and gradient values of all bins. To normalize the values of the bins we divide them by $2^n - 1$, where n is the number of bits representing the original input data. Figure 1 shows the effects of the preprocessing steps on the generated IGM histogram.

To calculate the clustering we use the *KMeans* algorithm from the *sklearn* library. The *KMeans* algorithm initializes the cluster centers within the given vector space and iteratively improves their positions by moving them according to the average position of the points closest to them. One downside of *KMeans* is the need to set the number of clusters manually. On the other hand the time complexity of the algorithm is one of its strong suits. The results it produces are also satisfactory for our use.

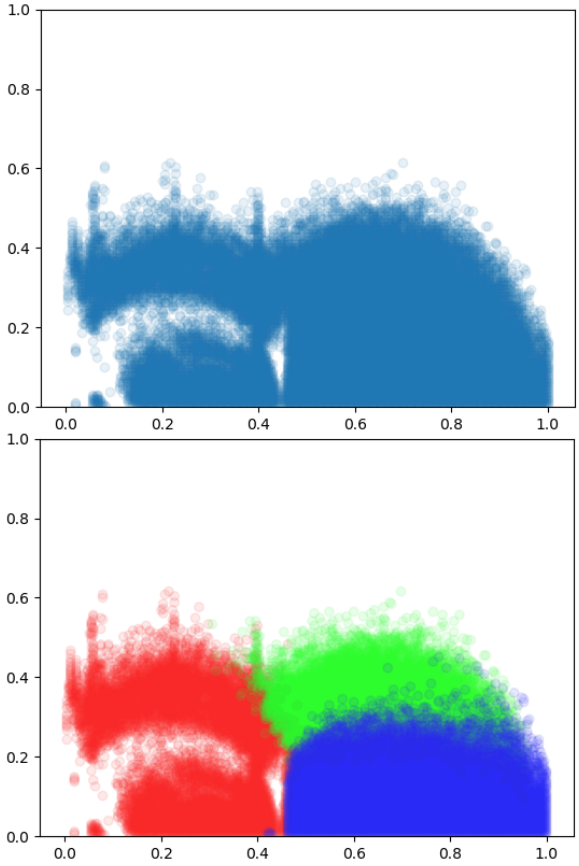

Figure 2: *IGM histogram generated by our implementation (top) and its clustering (bottom), where the number of clusters was set to three*

We perform the clustering using vectors containing five values. For each bin we use the gradient and intensity magnitudes as well as the mean position in all directions in the input data. That way we incorporate information regarding proximity in the IGM histogram as well as in the input data.

Figure 2 shows the IGM histogram and its clustering generated by our approach.

2.4. Transfer function generation

Given the data obtained by the clustering we wish to generate a transfer function appropriate for VPT. VPT presents the transfer function to the user as a collection of ellipses, where for each the user can manipulate its center, radius and color represented in the RGBA color model. Since the shapes of clusters can not be easily portrayed by ellipses we devised a naive approximation method.

We sample a $N \times N$ regular grid of intensity and gradient magnitudes in the range [0, 1]. For a given sample with indexes (i, j) we calculate the intensity and gradient of the sample as $I_i = \frac{i}{N}$ and $G_j = \frac{j}{N}$. We then check if there exist any bins that lie within a square with the center in (I_i, G_j) and line length of $\frac{1}{N}$. If such a bin exists we add a circle to the transfer function with the center in (I_i, G_j) and radius of $\frac{1}{N}$. The color of the circle is determined based on the cluster label of the bin found within the square.

As mentioned before VPT uses the RGBA color model. In our implementation we used the HSL color model to determine N most distant colors, where N is the number of clusters. The HSL values for a color with index i, were calculated by Equations [3,4,5], where U_n represents a random number from the interval [0, 1]. Finally we transform the colors from the HSL space to the RGBA space using the *colorsys* library.

After the transfer function is constructed a response is sent back to the client, containing the newly constructed transfer function. The transfer function is than downloaded as a *JSON* file, which the user can use within the VPT framework.

$$H_i = \frac{i * 360}{N} \tag{3}$$

$$S_i = 90 + U_n * 10 \tag{4}$$

$$L_i = 50 + U_n * 10 \tag{5}$$

Figure 3 shows an example of the clustered IGM histogram and its transfer function displayed by VPT, when sampling 50 points in the regular grid. Once thing to note is that due to the nature of *WebGL* the gradient values had to be inverted.

2.5. Transfer Function Manipulation

Additionally we implemented simple transfer function manipulation. Firstly we added a *Generate* button which generates five new transfer functions. These can than be selected for viewing by clicking on one of the five generated buttons. A *Select* button was also added, if the user wants to replace the current transfer function with one of the suggested ones.

To generate new transfer functions we took a simple approach. A transfer function is a collection of data points, with four spatial values and four color values. We simply iterate over all these data points and introduce small variations by adding random noise to all four spatial values.

3. Results

In the following section we present the results of our implementation. We focus on the time complexity of our approach and on the visual clarity of the renderings, when using transfer functions constructed by our approach.

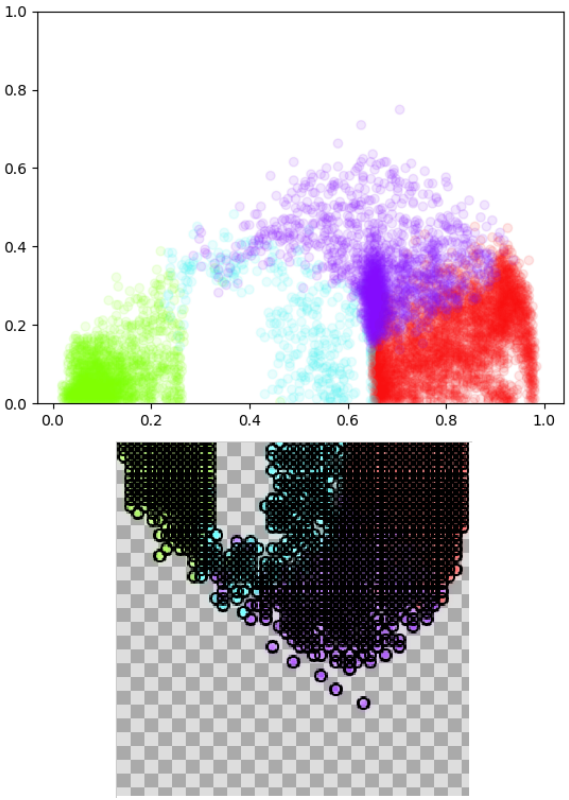

Figure 3: *Clustered IGM histogram generated by our implementation (top) and the transfer function built from the histogram as displayed in VPT (bottom)*

File Name	Size (x,y,z)	$Time_g$ (s)	$Time_c$ (s)
Aneurism	256,256,256	3.201	0.982
Baby	256,256,98	10.064	2.857
Bonsai	256,256,256	12.051	1.9461
Clouds	512,512,32	3.367	1.780
Tooth	103,94,161	4.344	2.022
Sheep	352,352,256	82.795	2.446
Ct_chest	384,384,240	32.984	1.649
Box	64,64,64	0.315	0.209
Daisy	192,180,168	2.506	2.083
Internals	128,128,227	1.264	0.940
Neghip	64,64,64	0.302	0.498
Nucleon	41,41,41	0.171	0.742
MRI_ventricles	256,256,124	21.263	2.408
Foot	256,256,256	15.596	2.555
Cross	64,64,64	0.106	0.188

Table 1: *Measured time complexity of fifteen different inputs*

3.1. Time complexity

We tested the time complexity of our approach on fifteen different inputs. We measured separately the time needed to calculate the bin data $Time_g$ and the time needed to preprocess and cluster the bin data $Time_c$. All results are shown in Table 1.

There is a clear correlation between the times $Time_g$ and $Time_c$ and the size of the input data. Time $Time_g$ seems to be much more sensitive to the size of the input data. This is a results of having to go over all points to calculate the bins and all of their values. Time $Time_c$ is less sensitive since we are dealing with bins, which are already a much more condensed representation of the input data, which we then further shrink with the preprocessing steps.

3.2. Visualization

We show examples of data visualization using VPT and transfer functions constructed using our implementation. Figures [4,5,6,7] show results for four different inputs. For each input we show the clustered IGM histogram (top-left), transfer function as shown in VPT (top-right) and the final rendering as shown in VPT (bottom).

Figure 4 shows the results obtained from the volumetric data of a babies head, using the Emission-absorption rendering model, with the *Steps* parameter set to 100 and the *Opacity* parameter set to 10. We set the number of clusters to three. From the visualization we see that the generated transfer function separates well the inside (red) and the edge (green) of the head. There appears to be some noise (blue) outside of the babies head at the bottom of the visualization, that was not removed by our implementation. Overall the visualization is clear, yet does not reveal many intricacies of the input data.

Figure 5 shows the results obtained from the volumetric data of a tooth, using the Emission-absorption rendering model, with the *Steps* parameter set to 100 and the *Opacity* parameter set to 10. We set the number of clusters to three. From the visualization we see that the generated transfer function separates well the edge (green) and the enamel (blue) of the tooth, as well as some anomalies (red) on the surface of the tooth that could correlate to imperfections on the surface of the tooth. The overall visualization is clear in showing the tooth and most of its defining features.

Figure 6 shows the results obtained from the volumetric data of internals of the human body, using the Emission-absorption rendering model, with the *Steps* parameter set to 100 and the *Opacity* parameter set to 10. We set the number of clusters to three. The transfer function separates the actual internals (green and red), as for the other clusters there is no clear interpretation. Visually the final rendering is not very clear and does not convey enough important information to the user. Many causes could be attributed to such a poor results, the most probable is that our preprocessing step removed many of the useful bins.

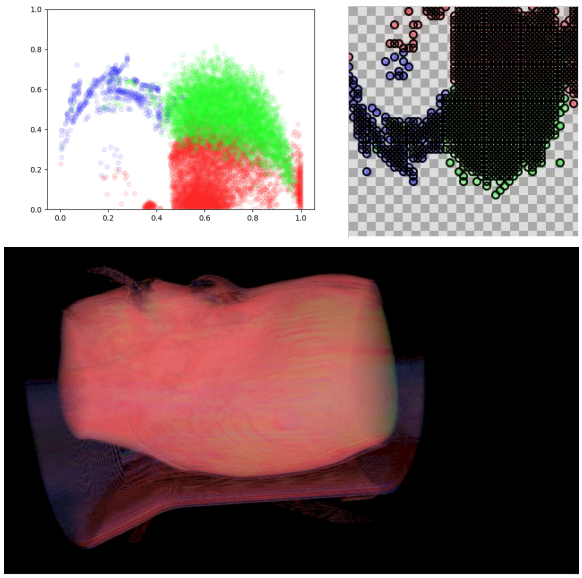

Figure 4: *Results for the input named baby*

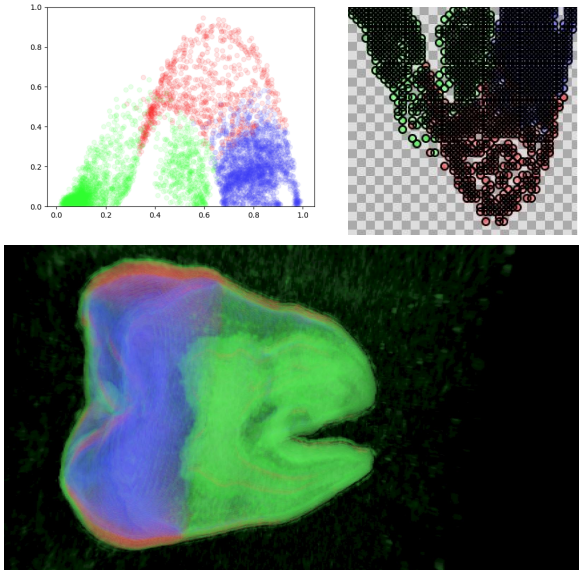

Figure 5: *Results for the input named tooth*

So far we have only shown results for the Emission-absorption rendering model, yet it is possible to also use the Multiple scattering rendering model. Figure 7 shows the results obtained from the volumetric data of an engine, using the Multiple scattering model, with the *Extinction* parameter set to 100, the *Scattering albedo* set to maximum, the parameter *Max bounces* set to 100 and the parameter *Steps* set to 100. We set the number of clusters to three. From the visualization we see that the transfer function separates the body of the engine (green) and the pistons and back-plate (red and blue). The blue cluster appears to be largely unnecessary appearing only on the edges of some parts. Overall the visualization is clear and conveys clearly the different materials used to construct the engine.

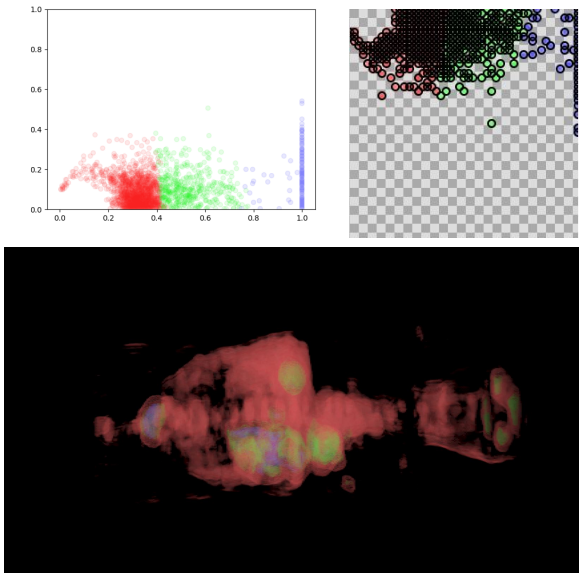

Figure 6: *Results for the input named internals*

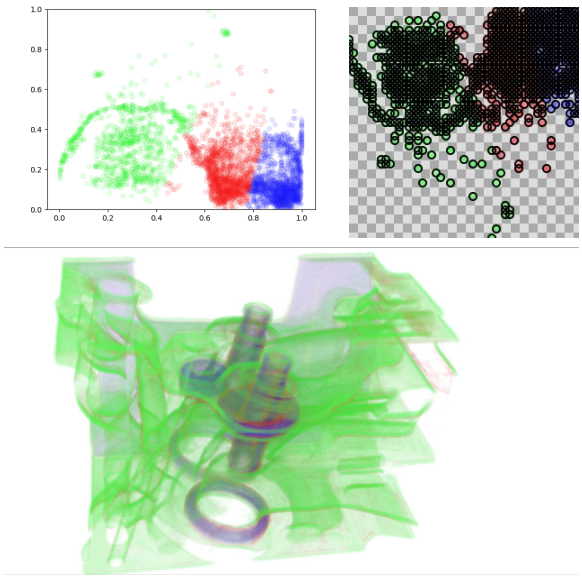

Figure 7: *Results for the input named engine*

Overall the renderings for most inputs are satisfactory, as different materials are clearly shown. However, the presented approach is sensitive to variation in the input data. All parameters regarding the normalization and preprocessing are not calculated individually for each input, we instead use hand-crafted approximated values for all inputs.

4. Conclusion

In the paper we present how we extended the functionality of the *Volumetric Path Tracing Framework* (VPT). Enabling the automatic generation of transfer functions of input data by using the IGM histogram clustering approach.

The implementation is divided into two parts. The first is calculation of the gradients, while the second is clustering and transfer function design. The gradient calculation step is implemented in a separate Python script, which calculates the gradients for a given *RAW* file and outputs a new file which the user can use in VPT. While uploading the input data into VPT the user can set the number of clusters. While loading the data on the client-side we calculate the bins and their positional statistics. Once all input points have been assigned to bins we send the data to the server side. Before clustering we perform some preprocessing steps, which remove noisy bins. To cluster the data we use the *KMeans* algorithm. Finally a transfer function is built from the clustering by sampling a regular grid over the intensity-gradient space of the bins. The transfer function is sent back to the client, where it is downloaded. The downloaded file can then be used in VPT.

The results of our implementation vary depending on the input data. In most cases the transfer functions made by our implementation are able to separate well the different materials present in the data. In some case it may occur that our preprocessing steps remove to many important bins, resulting in poor visualization.

5. Discussion

Our implementation overall performed well. For most inputs the resulting visualization, shows to the user intricacies otherwise difficult to see. That being said our implementation has a few drawbacks.

One of which is the time complexity of calculating the bins and their positional statistics. A possible improvement would be to send chunks of data to the server, which would then construct the bins in chunks. Such an approach would use more bandwidth, yet the data would be visualized sooner.

Another issue is the use of the *KMeans* clustering algorithm, which is not the most appropriate for the given data. Since we were concerned with the time complexity of the clustering we fell back to using *KMeans*, while other more appropriate clustering algorithms exist, such as *Affinity Propagation*.

Our approach can sometimes fail to construct a good transfer function. The problem mainly lies in the preprocessing stage, where we remove noisy bins. One possible solution would be to have soft bounds defined by the input data itself, based on which we would remove the bins. As it stands the conditions implemented are sometimes to strict while other times they are to lenient.

References

[ZYZ*16] ZHANG T., YI Z., ZHENG J., LIU D. C., PANG W.-M., WANG Q., QIN J.: A clustering-based automatic transfer function design for volume visualization. *Mathematical Problems in Engineering 2016* (2016). 1

[BM18] ŽIGA LESAR, BOHAK C., MAROLT M.: Real-time interactive platform-agnostic volumetric path tracing in webgl 2.0. In *Web3D 2018 : proceedings* (2018), pp. 1–7. 1