Statistical Nonparametric Estimation and Classification Problems in 3D Rendering and Image Processing

Bu Zhou

Problem (I) 3D Reconstruction

One popular topic in computer graphics is to reconstruct 3D environment from 2D images. To make it clear, a 3D rendering algorithm for the proposed problems will be explained first. The basic idea of 3D (voxel) rendering, is kind of like mapping 3D points with colors, or voxels ([1], [12]), to 2D pixels on the view plane. The view plane can be thought of as a camera or computer screen; its content is the rendered result, generally a 2D image. The simplest rendering procedure is the ray marching voxel algorithm ([2], [3], Figure 1 and 2): for each pixel on the view plane, trace a ray, and march the ray in the 3D voxel space, until the ray hit a voxel; then we extract the color value of the hit voxel and write it to the screen. This is the so-called "first-hit" test. The rendering process is actually a simple searching algorithm with a 3D array, VoxelData[x][y][z], where the index (x, y, z) is the position of the voxel in 3D space and the value of the array member is the color for that voxel.

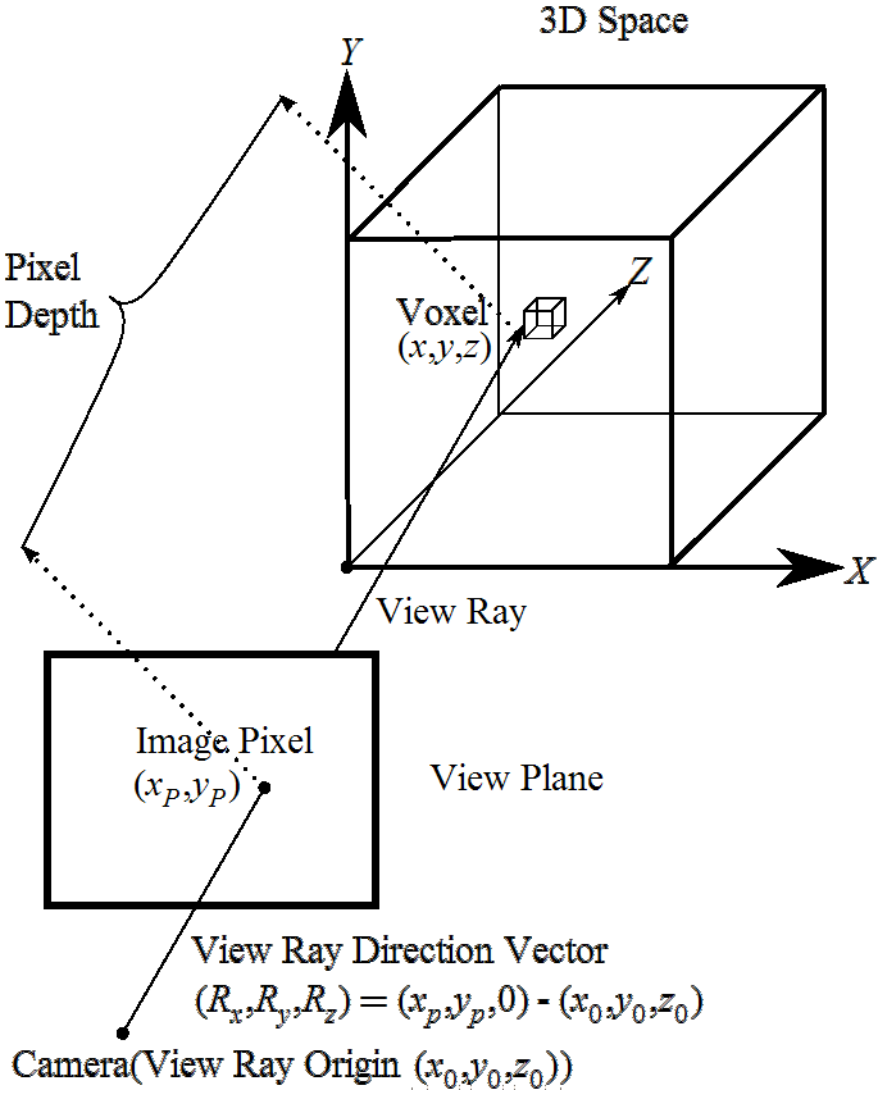

Figure 1. Ray Marching Voxel Algorithm

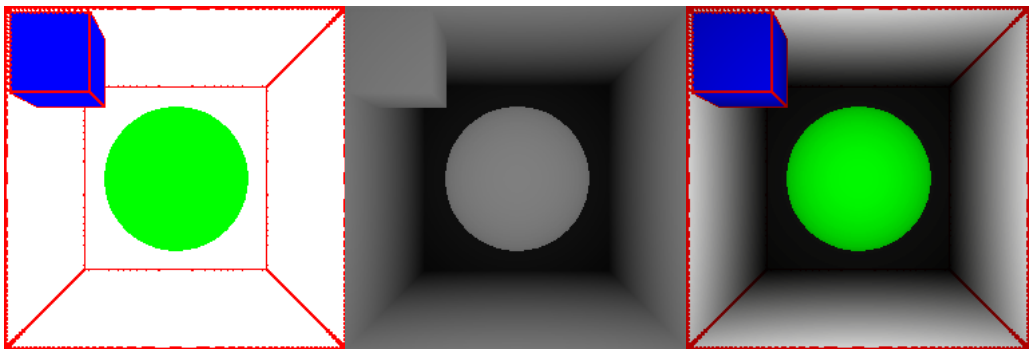

Figure 2. Examples of Ray Marching Voxel Algorithm
A cube and a sphere in a box.
Left: rendered without light falloff ([3]);
Middle: depth image;
Right: rendered with light falloff ([5]).

For 3D reconstruction, we want to find the 3D position (x,y,z) for every pixel on the 2D view plane. The first step is to figure out the depth (Figure 2, Middle) of those 2D pixels, or more precisely, the distance of the voxels to the view plane in the corresponding view ray's direction. Once we have the pixel's depth information, the 3D position can be obtained via some basic vector calculations. The depth image can also be used to create some pseudo 3D effects ([4]). When estimating the depth, the pixel value - color is the only information we can obtain, so we need a model describing the relationship between color and depth of the pixels. A model for this relation can be written as $g(color) = f(distance)$, where g is a function of the pixel value, such as brightness, hue and saturation; f is a function of pixel depth. For example, a physics model for light transmitting is the distance falloff mode ([5], [6]): light intensity = f(distance). f is the function to describe how the light intensity weakens with the distance, i.e., objects further away will look darker. In real life, complex physics models are needed to design the function f, which may take into consideration many factors in light transmitting, such as light scattering, absorption ([12]). In rendering, we can directly choose different distance falloff functions f to create different visual effects, while in depth estimation, we do the reverse thing - we need to estimate the unknown f using the data generated from the image pixels. In Statistics, we can view it as a nonparametric estimation problem.

Problem (2) Shadow Rendering

After we rendered the voxel data to a 2D image, a natural thing is to add some other enhancing effects, such as shadows, in order to make the result more realistic. A simple way to add shadows is the back tracing algorithm ([7], Figure 3 and 4). For each 2D pixel in the image, find its position (x, y, z) in the 3D space (we can record the position for every pixel in its rendering process; the position can also be calculated easily if we know the pixel's depth and some basic rendering parameters, as mentioned in problem (I)), cast a ray from the voxel (x, y, z) to the light source in the 3D environment. If the ray reached the light source without hitting any other

voxels, the voxel and hence its image pixel projected on the view plane is out of the shadow area, otherwise, they're in the shadows.

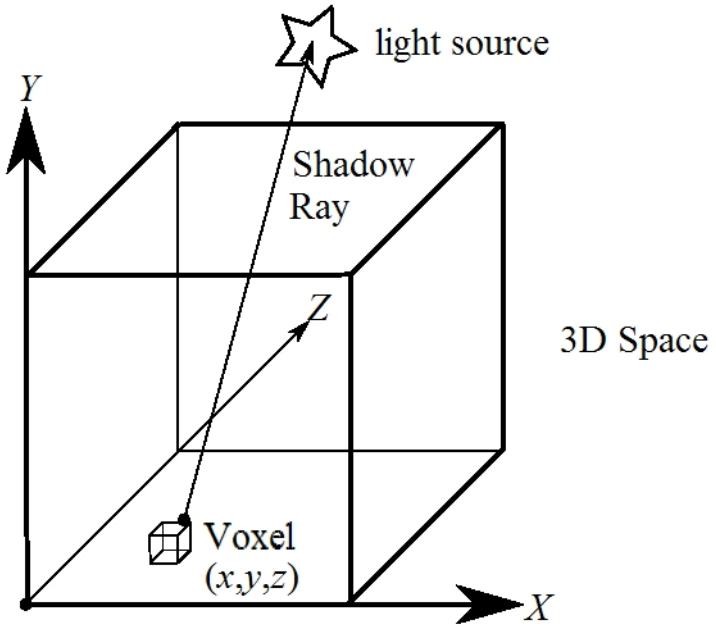

Figure 3. The Back Tracing Algorithm for Shadow Rendering

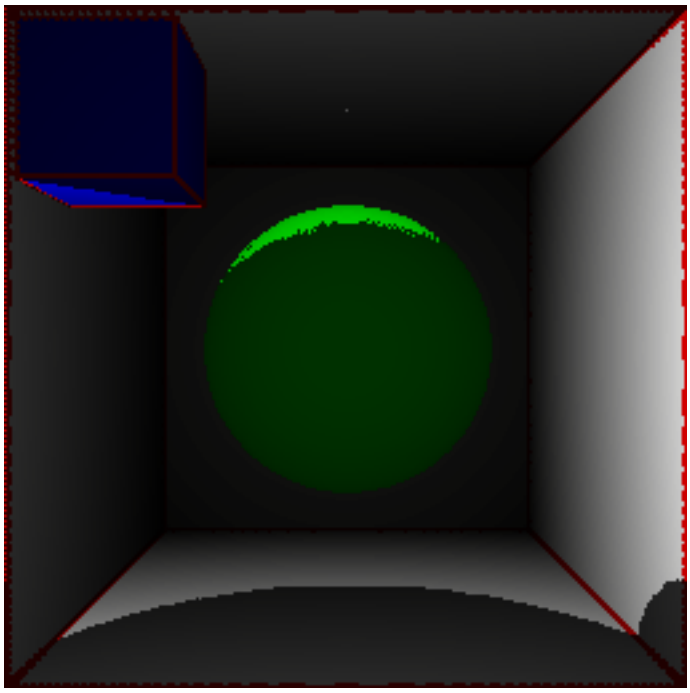

Figure 4. Example of the Back Tracing Algorithm for Shadow Rendering ([7])

However, this naive algorithm is computing intensive if the resolution of the 2D image is large. An alternative way is to sample some discrete 2D pixels for shadow rendering and use the sampled data to approximate the shadow areas. It is therefore a problem of classification - we now need to find the separating hyperplane or decision boundary between the areas in shadow and areas not. How much data do we need to reconstruct the shadow areas? What about the performance and computing complexity compared to full rendering? For a 2D image of $w \times h$ pixels, the back tracing algorithm for rendering shadows requires more than $O(42 \times w \times h)$ calculations. Treat the shadow rendering problem as a statistical classification problem can give a chance to find some better shadow rendering algorithm because the new algorithm may only depend on a small subset of the full $w \times h$ pixels.

Extension and Application:

If we consider the previous 3D problems solely in 2D, a general concept is image post-processing. Given an input image, we can use a transformation function on the pixel values, called a filter, to produce some stylized images. Figure 4 is an example of cartoonizing image ([8], [13]): picture on the right is generated by applying a series of post-processing procedures to the photo on the left. Use function f to represent a filter, we have (new pixel value)= f(original pixel values). Note that f can be the distance falloff function in problem (I), f can also be a function of more than one variables here, i.e., when the new pixel value is determined by both its original value and its 8 neighbor pixels.

Figure 5. A Cartoonized Photo ([8])

An interesting problem here is to estimate the filter function f, use some reference images ([13]). The two pictures in Figure 5 can be used as example data for this nonparametric estimation problem. The objective here is either to simulate some artistic style (estimate unknown f) or to find some faster algorithm for post-processing (find a function f which can nicely approximate a series of known post-processing procedures).

Statistical Challenges in the Problems - Spline Tools as an Example

For nonparametric estimation of the function f, splines are popular tools. However, several technical issues need to be considered when using them.

1. Generalization of spline smoothing and their asymptotic properties

The theory of regression splines and smoothing splines are well established ([14], [15]). However, only a few asymptotic properties ([16]) are known about a popular kind of splines called penalized splines. Penalized splines are first proposed by Eilers, Marx ([17]) and are fully discussed by Ruppert, Wand, and Carroll ([18]). Penalized splines are generalization of regression splines and smoothing splines, and they are widely used in applications because of their better performance.

Egerstedt and Martin ([19], [20]) studied splines smoothing in a control theory's view, as a problem of optimization in vector space. They proved the convergence of generalized splines in a unified and elegant way, but the rate of convergence is not given.

The first challenge here is to choose a specific kind of splines, or other tools such as local polynomials ([21]), support vector machines ([22], [23], [24]), and neural networks ([24], [25]), which are suitable for the problems both in theory and application.

2. Model selection of spline knots

Another practical concern for regression splines and penalized splines is how to select the number and locations of spline knots. For selecting the number of knots, Ruppert ([26]) gave his suggestions based on simulations using GCV. For the locations, a common practice is to place the knots on a grid of equally-spaced sample quantiles. However, a clear and objective criterion is still needed. Rissanen ([27], [28]) developed the MDL principle as a general principle for model selection. Lee ([29]) and Liski ([30], [31]) proposed to use MDL principle to choose the knots and reported their simulation results. However, no related theoretical results such as the consistency of their model selection procedure can be found.

3. Nonparametric or semiparametric?

Besides fully nonparametric solutions to these problems, another approach is to use semiparametric or even parametric models. Take the depth estimation problem as an example, we may refer to some physical models and make it a semiparametric or parametric estimation problem. The advantage of parametric models is the explainability. Since parametric models may not be very useful in practice, I personally prefer the nonparametric solution.

The Pipeline for Real Life Applications and Statistical Simulations

For statistical estimation, the GNU-R system ([9]) is suggested. 3D rendering can be done in a C program using the SDL library ([10]). Data for simulation will then be generated and supplied to R. For real life application, digital photos took by cameras can be used as the data. The work flow is showed in Figure 6, and a simple proof-of-concept R & C implementation of the pipeline is available online ([11]).

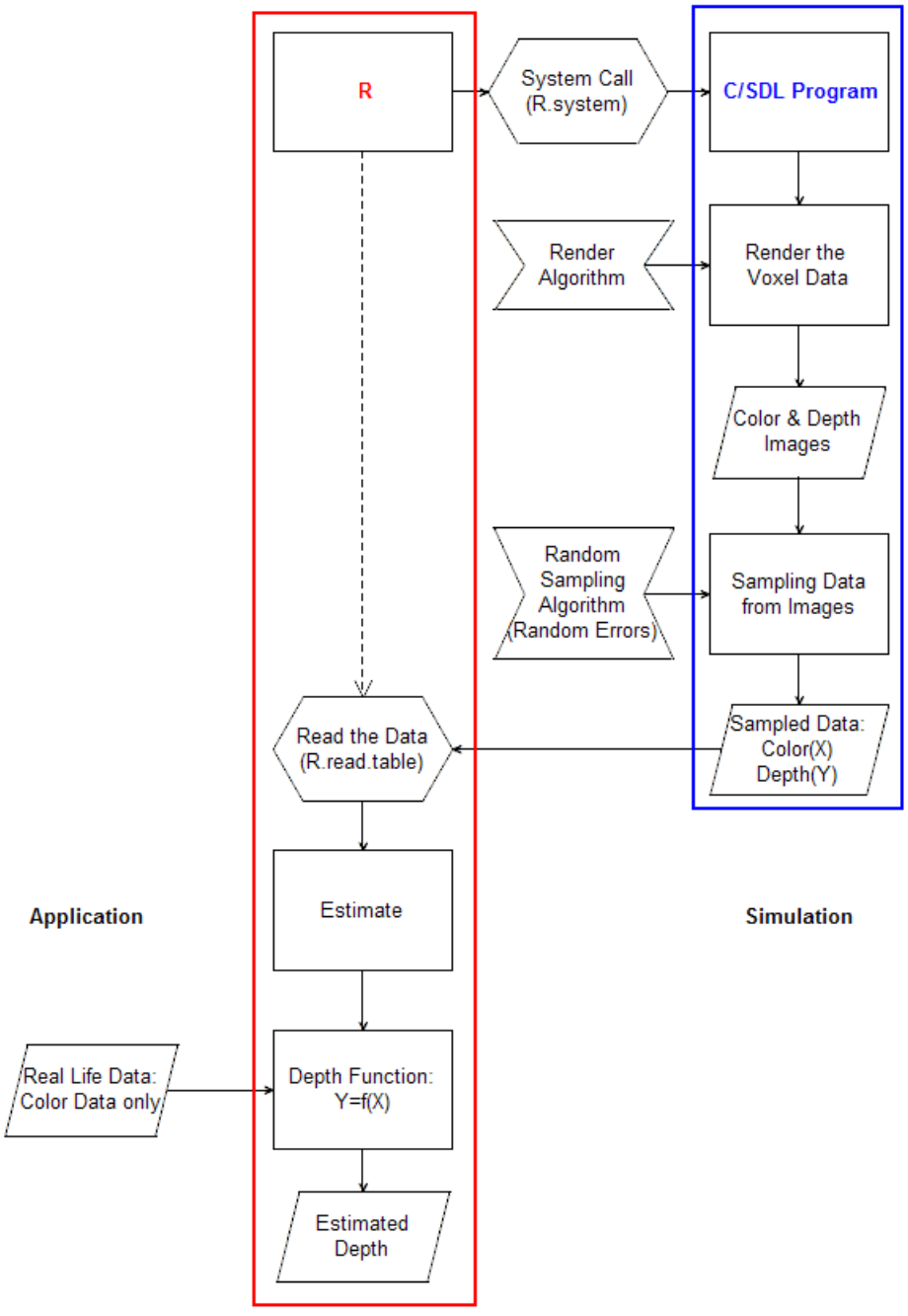

Figure 6. The Pipeline for Depth Estimation

Main References:

Online Resources:

[1] Voxel(volumetric pixel):

http://en.wikipedia.org/wiki/Voxel

[2] Ray Tracing:

http://en.wikipedia.org/wiki/Ray_tracing_%28graphics%29

[3] AS3 Implementation of Ray Marching Voxel Algorithm:

http://bruce-lab.blogspot.com/2011/08/simple-ray-tracing-voxel-demo.html

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/Demo_trace.swf

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/RaytracingVoxel.as

[4] Pseudo 3D Effects using Depth Image -

3D Background:

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/Pseudo3D/Background3D/

Parallax Mapping:

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/Pseudo3D/ParallaxMapping/

[5] Distance Falloff:

http://en.wikipedia.org/wiki/Shading

[6] AS3 Implementation of Light Falloff:

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/Demo_light.swf

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/RaytracingVoxel_LF.as

[7] AS3 Implementation of Back Tracing Algorithm for Shadow Rendering:

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/Demo_shadow.swf

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/FlashDemo/RaytracingVoxel_Shadow.as

[8] Cartoonize as Post-Processing:

http://bruce-lab.blogspot.com/2009/04/cartoonize-image-by-as30.html

[9] R: Statistical Computing Language

http://www.r-project.org/

[10] SDL: Simple DirectMedia Layer

http://www.libsdl.org/

[11] R & C Implementation of the Proposed Pipeline:

http://bzstat.googlecode.com/svn/trunk/DepthEstimation/CRPipeline/

Articles and Books:

[12] Real-Time Volume Graphics, Klaus Engel, Markus Hadwiger, Joe Kniss, Christof Rezk-Salama, Daniel Weiskopf, A K Peters, 2006

[13] The Algorithms and Principles of Non-photorealistic Graphics: Artistic Rendering and Cartoon Animation, Weidong Geng, Springer, 2010

[14] Local Asymptotics for Regression Splines and Confidence Regions, X. Shen, D.A. Wolfe, S. Zhou, Ann. Statist, 1998

[15] Nonparametric Regression and Spline Smoothing, Second Edition, Randall L. Eubank, CRC Press, 1999

[16] Asymptotic Properties of Penalized Spline Estimators, Gerda Claeskens, Tatyana Krivobokova, Jean D. Opsomer, Biometrika, 2009

[17] Flexible Smoothing with B-splines and Penalties, Eilers, Marx, Statistical Science, 1996

[18] Semiparametric Regression, David Ruppert, M. P. Wand, R. J. Carroll, Cambridge, 2003

[19] Control Theoretic Splines: Optimal Control, Statistics, and Path Planning, Magnus Egerstedt, Clyde Martin, Princeton, 2009

[20] Control Theoretic Smoothing Splines, S. Sun, M.B. Egerstedt, C.F. Martin, Automatic Control, IEEE Transactions on, 2000

[21] Local Polynomial Modelling and Its Applications, Jianqing Fan, Irene Gijbels, Chapman and Hall, 1996

[22] A Tutorial on v-support Vector Machines, P. Chen, C. Lin, B. Schölkopf, Applied Stochastic Models in Business and Industry, 2005

[23] The Nature of Statistical Learning Theory, Vladimir Naumovich Vapnik, Springer, 2000

[24] The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Trevor Hastie, Robert Tibshirani, Jerome Friedman, Springer, 2009

[25] A Statistical Approach to Neural Networks for Pattern Recognition, Robert A. Dunne, Wiley, 2007

[26] Selecting the Number of Knots For Penalized Splines, David Ruppert, 2000

[27] A Tutorial Introduction to the Minimum Description Length Principle, Peter Grünwald, Advances in Minimum Description Length: Theory and Application, MIT, 2005

[28] Information and Complexity in Statistical Modeling, Jorma Rissanen, Springer, 2007

[29] Regression Spline Smoothing using the Minimum Description Length Principle, Thomas C. M. Lee, Statistics & Probability Letters, 2000

[30] MDL Knot Selection for Penalized Splines, Antti Liski, Erkki P. Liski, WITMSE, 2008

[31] Model Selection in Linear Mixed Models using MDL Criterion with an Application to Spline Smoothing, Erkki P. Liski, Antti Liski, WITMSE, 2008