点群データの擬似サーフェースレンダリング

川田 弘明　　金井 崇

慶應義塾大学 環境情報学部
E-mail: { t02282hk/kanai }@sfc.keio.ac.jp

1. はじめに

点群によるレンダリングでは、面を構成するデータ等を必要とせずに物体をレンダリングする事ができる[1,2]。この際、点群のみによるデータではレンダリングを行った際にどのようにして点と点の間の隙間を埋めるのかという事が問題となる。

本稿では、点群を 2 次元に投影した状態から面を生成する事を基本とした擬似サーフェースレンダリング手法について提案する。また、点群のみのデータからシェーディングの処理もリアルタイムに行うために、シェーディングの計算に必要なデータも同時に計算し表示する。本手法は、特に三次元測定器により得られる点群データのラフチェック等に有効である。

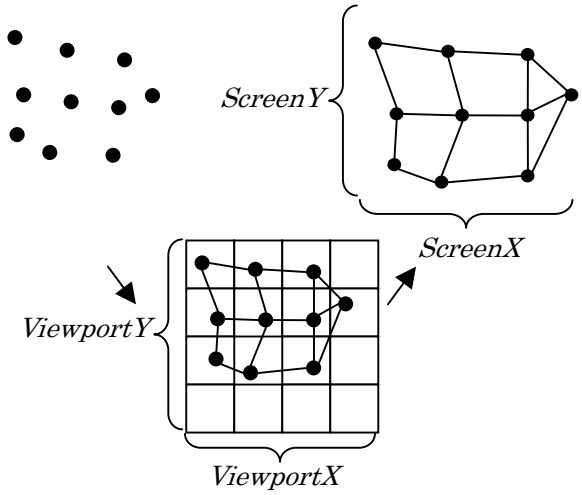


図 1: レンダリングの手順

2. 提案方法

2.1 レンダリング手法の手順

本提案方法の全体のレンダリングの基本的な流れについて説明する(図 1)。

本提案手法では、

1. 投影変換による 2 次元座標
2. 対応する三次元座標
3. 視点から一番近い点
4. 法線
5. 色情報
6. 点の数

を蓄えておくためのバッファがそれぞれ必要となる。

まず、実際にレンダリングする画像の大きさよりも小さいバッファを用意する。それぞれの 3 次元の座標の頂点において投影変換を行い、計算される値 2 次元座標をこの小さいバッファに格納する。小さいバッファのビューポートの大きさ(*ViewportX, ViewportY*)は式(1)のように決定する。*ScreenX*、*ScreenY* はそれぞれレンダリングする画面の大きさを示している。ここで *S* は画角、点の密度、スクリーンの大きさによって決定する。

$$ViewportX = ScreenX / S$$
$$ViewportY = ScreenY / S \qquad (1)$$

2 次元座標はラスタライズ時に格納する。また、同時に視点から一番近い点を記録しておくバッファ、3 次元の座標を記録しておくバッファ、色を記録しておくバッファにそれぞれ値を記録する。これらのバッファの位置に複数の点を記録するときには、それらの値の平均をとる必要がある。平均値を計算する場合は、記録された点の数を示すバッファを使用し、全ての頂点についての処理を終った後に行う。

次に 3 次元の座標を記録しておくバッファから法線を生成し、これらの法線を計算した法線を記録しておくバッファに記録する。

次に、*S* 倍に拡大させライティングの計算を面が構成できる点ごとに計算し表示する。ライティングの計算は法線を記録したバッファの法線のデータを利用して行う。

最後に図 2 のように面を生成する。隣接するピクセルを参照し、3 つの点が存在すれば四角形を描く事ができ、2 つの点が存在するのであれば三角形を描く事ができる。また、それ以外の場合は面を生成しない。

2.2 画像の劣化の改善

2.1 において、画像を拡大する時に、縮小した状態での解像度のデータから拡大すると画像が粗くなってしまう。そこで、2 次元に投影した状態において小数点以下の値も保持しておき拡大する時にはこの値を使用する。この値によって元の大きさに拡大したときに形状の劣化を多少

防ぐ事ができる。また、点が複数同じバッファの位置に存在する場合は、そのバッファの点の値の平均値をとる事にする。また、平均値をとるための対象となる点は視点からスキャンされたデータの距離により選択する。実際にもとの解像度で描画する際に、このバッファから面を生成し拡大して描画する。面を生成する際には対象とするバッファの位置の付近に点が存在する場合に面を生成するようにする。

2.3 法線情報の生成

スキャンされたデータ等において法線情報を持っていない場合は、法線情報を生成する必要がある。本手法では投影した点に対してその付近の点から法線を生成する。実際には、2 次元に投影すると同時に、その点に対応する元となる 3 次元空間での座標もバッファに記録しておき、このデータを使い法線情報を生成することにした。また、対象とするバッファの位置の付近の点から生成される面の法線を平均したものを法線とするが、法線を生成する面の頂点が中心の頂点に比べ遠すぎる場合には、この頂点は法線を生成する点の対象にはしない。これらの距離は、スキャンされたデータに応じたものを使用する。

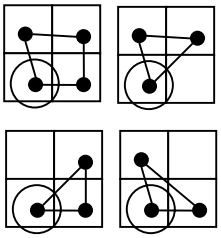


図 2: 面の生成

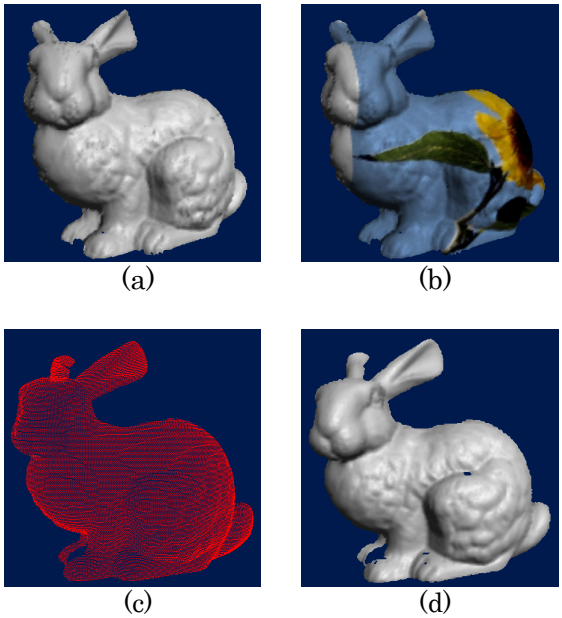
3. 結果と議論

本手法によるレンダリング結果を示す。計算環境は、PentiumIII 800MHz、ビデオチップは Mobile Radeon を用いた。例題としてスキャンされた Stanford Bunny のデータを使用した。図 3 はそれぞれ 4 つの方向からスキャンしたデータを重ねた結果(142587 頂点)をレンダリングした場合であり、(a)は灰色のみの色で描画した結果、(b)は 2 次元の画像によって色情報を与えて描画した結果となっている。

また、図 3 の(c)の画像は 1 方向からスキャンされたデータ(40256 頂点)を 1 ピクセルの点で描画したものであり、(d)の方は本手法による結果となっている。

計算時間は、図 3 の(a)、(b)が 0.45 秒、(d)は 0.31 秒となっている。今回の実装では、ライティングの計算はソフトウェアによる計算となっている。

次に、本手法による問題点を示す。まず、問題点としてエッジが上手くレンダリングできない事がある。図 4.2 の(a)では、後ろに物体が無い場合だが、エッジがギザギザに描画されてしまっている。また、図 4.2 の(b)では、後ろに物体がある部分のエッジがぼやけてしまうという問題がある。



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

図 3 レンダリング結果

(a) (b)

図 4 問題点

4. おわりに

本稿では点群のデータを 2 次元に投影した状態から面を生成し、レンダリングを行う手法を提案した。今後しかし、本手法では細かい部分のレンダリング、点と点が重なる部分、視点から遠い部分のレンダリングが不得意である。今後はこれらの部分においての改善を行っていく必要があると考えられる。

5. 参考文献

[1] Szymon Rusinkiewicz and Marc Levoy: “QSplat: A MultiResoliton Point Rendering System for Large Meshs”, Proc. SIGGRAPH 2000

[2] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar and Markus Gross: “Surfels: Surface Elements as Rendering Primitives”, Proc. SIGGRAPH 2000