法線情報のみを用いたパーティクルシミュレーションのデザイン

Designing Particle Simulation by Normal Information

大和田 茂　藤木 淳*

Summary. 本稿では二次元画像を用いた疑似三次元的なパーティクルシミュレーション方法と, それを実現するユーザーインターフェースを提案する. 我々の手法では, ユーザーが表現したいシーンは二次元画像の各ピクセルに,RGB の色情報に加えてその点の三次元の面の方向 (法線) を付加した法線マップとして表現される. 法線マップは二次元表現のフレキシブルさと三次元的効果のリアリティを兼ね備えた一般的なシーン表現であるが, それを積分することにより実際に三次元形状を再構築できるケースは限られているため, シミュレーションなどの処理を組みこむことが難しかった. 我々の方法では三次元形状を再構築せず, 法線情報のみを直接物理計算に用いるため, 大局的な一貫性のない法線マップを持ったシーンにおいてもパーティクルを用いた簡単なシミュレーション実行が可能である.

1　はじめに

コンピュータグラフィックスにおいては, 大まかに言って形状の三次元的な位置情報よりもその微分情報の方が重要であると言える. 例えば, 人間はシーン内の奥行きを二次元ディスプレイから知覚することはできないが, 面の方向情報なら陰影や変形などから簡単に知覚することができるのであり, それゆえ, 例えば CG においては陰影づけや形状変形のために法線やラプラシアンなどの微分情報を用いることが非常に効果的である [1, 8]. また, そういった情報は三次元の形状そのものを扱うよりもかなり容易かつ自由度が高いので, とりわけ近年では二次元画像中に定義されている微分情報や輝度勾配を操作し, 利用する研究が増加してきている [3, 4, 5, 6].

我々は, 二次元画像の各ピクセルに定義された法線情報を直接用いて, あたかも三次元空間でパーティクルが反射しているかのようなシミュレーションを行うことを試みる. 法線情報を直接用いる利点は, シーンの三次元的な一貫性に注意を払わなくてよいという点である. 法線情報があれば, それを積分すれば三次元の形状を再構築できると考えられるかもしれないが, 実はそれは常に可能なわけではない (例えば二次元画像上で A というピクセルから B というピクセルまでの一次元の経路を考え, 異なる経路が異なる積分結果を与えるような矛盾した法線情報を考えることは難しくない). そのため, 三次元モデリングのインターフェースとして法線情報を用いる場合には制約が大きくなり, 様々な工夫が必要になることが多い [3, 2, 7]. このことは, 法線情報の方が三次元形状をそのまま作成するよりも大きな自由度を持っている, と言いかえることもできる.

* Shigeru Owada, ソニーコンピュータサイエンス研究所, Jun Fujiki, 日本学術振興会

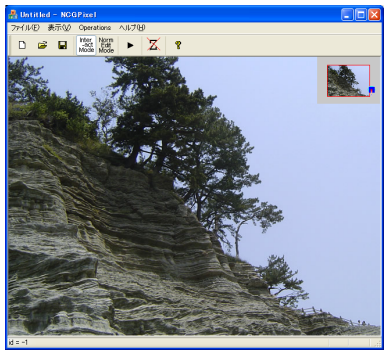


図 1. 画像をロードしてプログラムをスタートする.

我々のアプローチでは, 法線から三次元シーンを構築しようとせず, 法線をそのまま直接シミュレーションに用いることができる. シミュレーション自体は比較的簡単なものに限られるが, 三次元的な一貫性に心を煩わされることなくシーンをデザインできるという大きな利点を持っている.

2　ユーザーインターフェース

提案システムでのシミュレーションデザインは, まず通常の二次元画像 (各ピクセルが RGB の色情報を持つ) を読みこむところからはじまる. これは画像をメインのウィンドウにドロップすればよい (図 1).

次に, 法線エディットモードに入り, ユーザーがシミュレーションに使いたいピクセルに法線情報を与える. このモードでは, 元の画像の色情報が薄く半透明表示されている (図 2a). 法線の向きは, 色で表現されているほか, 右上のプレビューウィンドウで三次元的に確認することもできる. 法線情報をエディットするには, 左ドラッグで領域をセレクトし, その結果現れるハンドルをドラッグして与える方法 (図 2b-d) と,

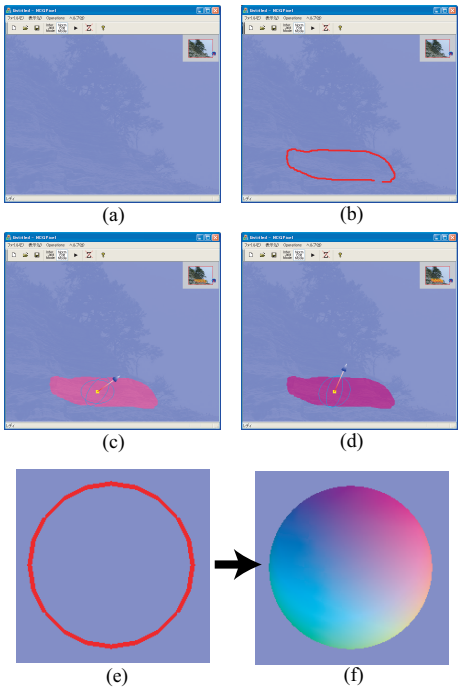
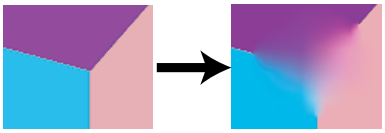


図 2. 法線のエディット


平滑化フィルター

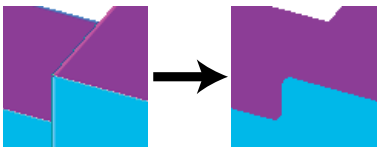

多数決フィルター

図 3. 法線に対するフィルターオペレーション

シフトキー押下とドラッグにより球体状の法線を描く方法 (図 2e-f) の二通りがある. また, すでに描かれている法線を, 画像フィルタにより加工する機能もある. 実装されているフィルタは, 現在ではスムージングフィルタと多数決フィルタの二つである (図 3).

法線デザインが終わったら, シミュレーションをデザインする. 最初に三次元的な重力方向を決める. 画面右下に重力方向が示されている. このハンドルをドラッグすれば重力の向きが変更できる (図 4a). また, 画面内のピクセルに定義されている法線を指定して, 重力をその逆方向に強制的に指定する方法もある (図 4b). 後者の方法は, シーン内に水平面が存在する場合に有効である.

次はパーティクルの湧き出し口を指定する. これには, 画面内の任意の位置でクリックして出現するポップアップメニューから選べばよい. デフォルトではパーティクルの初速度は, その点の法線方向に一致している (図 5a). この初速度の大きさと向きは, やはりハンドルをドラッグ操作することにより変更できる (図 5a-c). 湧き出し口を指定したら, あとはツールバーの再生ボタンを押せばシミュレーションが開始される (図 5d). 最初の段階ではパーティクルは放物線を描き, そのまま画面外へと落ちてゆく.

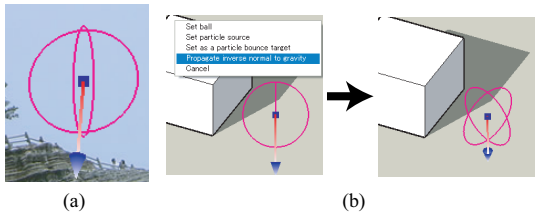


図 4. 重力方向のエディット

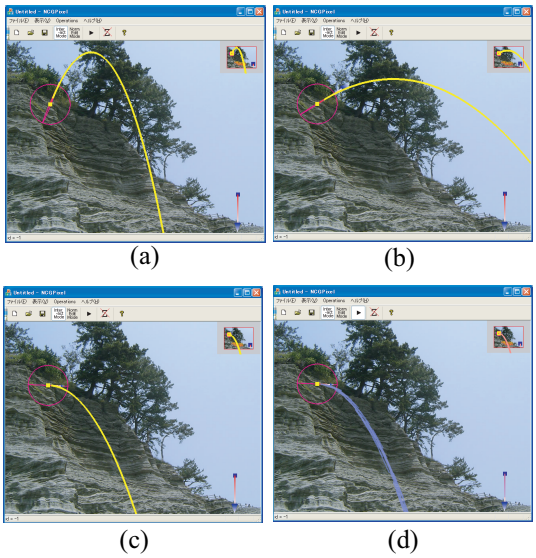


図 5. パーティクル動作の指定

次に, パーティクルが反射すべき地点をマウスストロークで囲って指定する (図 6a). すると, 湧き出し口から出てきたパーティクルはこの局所的に作られた面上で反射することになる (図 6b,c). 反射した時にパーティクルの (三次元的な) 速度ベクトルが変更されるが, この新たな速度は, 法線情報を用いて計算される (モーメントと摩擦を考慮しないシミュレーションでは, 衝突により向きを変更する運動量は衝突面の法線方向にのみ働く). この反射領域の指定は, ユーザーが望む回数だけ行うことができる (ただし, 反射が一度起こるたびにエネルギーが失われ, 跳ね返り度合いが小さくなることは考慮する必要がある).

法線情報はスクラッチから作成するのではなく, 既存の三次元形状をレンダリングしたものから得る

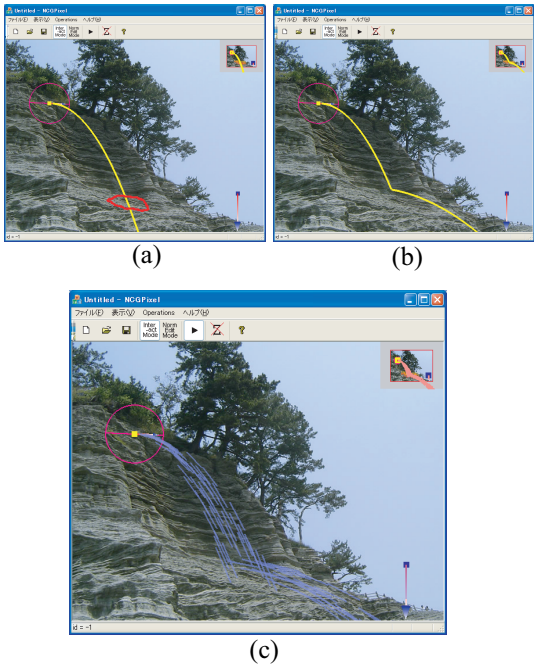


図 6. パーティクルの反射位置の指定

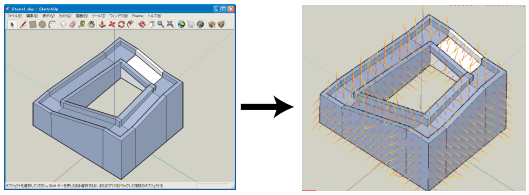


図 7. OpenGL からの法線情報取得

ことも可能である．これには GLIntercept と呼ばれる,OpenGL のシステム dll を置き替えることにより任意の OpenGL プログラムのグラフィックスシステムにアクセスできるプログラムを用いて, 外部プログラムの Z バッファを得ることにより行えるようにした. この Z バッファ情報を微分すれば, すぐに法線情報を得ることができる. 図 7 は, これにより Google SketchUp の Z バッファ(と色バッファ) を得た様子である．この場合, 法線情報は三次元的な一貫性を持つが, 法線エディタで少しエディットすると一貫性を失ってしまう.

また, 現在では実装されていないが, 入力画像を写真とする場合には画像処理分野で研究されてきた “Shape from Shading” と呼ばれる一連の三次元再構築手法を用いて法線を計算することも可能であると考えられる [9].

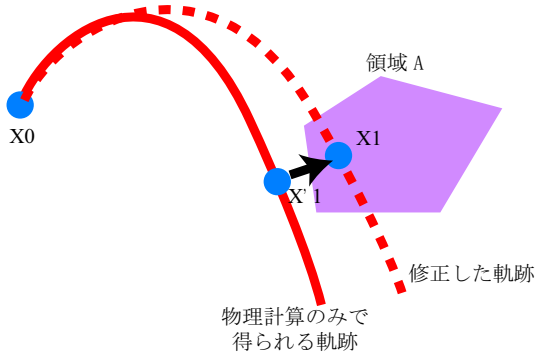


図 8. シミュレーション計算

3 実装

本システムの実装は非常に簡単である. まず, 各パーティクルはユーザーが指定したピクセル以外の物体とは干渉しない点とする．従って, このシミュレーションは, 衝突がない限りは各時間ステップにおいて速度ベクトルを足しこみ, 速度ベクトルには重力加速度を足しこめばよい. 結果は三次元空間内において放物線となる. 我々のシミュレーションでは奥行き方向 (Z 軸方向) の位置情報を用いないので, パーティクル位置は二次元ベクトルとして表される. ただし, 速度ベクトルは三次元ベクトルである.

物体との衝突は, 空間内での接触判定ではなく, ユーザーによって指定されたターゲットピクセルの位置情報を元に算出された衝突時間によって規定される. 具体的には, パーティクルが最初にシーン内に現れた時点や衝突が起こった時点で, 次の衝突が起こる時間を計算しておき, それをパーティクルごとに記録しておく. パーティクルが最初にシーン内に現れた時点, および衝突が起こった時点の時間を t_0 とおき, その直後の位置ベクトルを $X_0 \in \Re^2$, 速度ベクトルを $V_0 \in \Re^3$ とする. そのパーティクルが次に衝突を起こす場合, そのターゲットは二次元の領域としてユーザーによって定義されているはずである. この領域を $A \subset \Re^2$ と置く (図 8). 実際に現在注目しているパーティクルが衝突する点は,A のうちのどれか一点であるので, この点を A からランダムに選び, その位置を $X_1 \in A$ とする. 従って, パーティクル集合はこれから衝突する領域の広さに応じて拡散した動きを見せることになる.

こうしておいて, パーティクルが t_0 以降に描く二次元の軌跡を計算し, その軌跡の中で最も X_1 に近い点を選び, この位置を $X_1' \in \Re^2$, ここに到達する時間を t_1 とする. X_1 と X_1' は完全に一致するわけではないので, この差分をパーティクルの軌跡に分散させて, 完全一致するようにする (図 8). すなわち, 実際にシミュレーションを行うさいには, 各時間ステップで

計算された正しい位置に, $\frac{X_1'-X_1}{t_1-t_0}$ を足しこんでいくものとする.

衝突そのものには摩擦やトルクの影響がないものとする. 従って, 衝突前後の速度ベクトルをそれぞれ V,V' とし, 正規化された衝突面での法線を N とおくと, $V'=V-(1+\mu)(V\cdot N)N$ となる. μ はどの程度弾性的に反射するかを決める, 0〜1 の間のユーザーが与えるスカラー定数である.

4 結果

提案システムを用いてデザインしたシーンの例を示す. 図 9 は幾何学的なシーンで, パーティクルが階段を落ちていく様子である. 既存手法では三次元シーンを作り, ユーザーが意図した (二次元的な) 位置でパーティクルが反射するような初期状態をシミュレータに与えなければいけないので非常に手間がかかるが, 我々のシステムでは元々二次元的な情報だけを用いているので非常に簡単に意図した結果を得ることができる. また, 途中の段の幅が大きくなっているが, 処理を三次元で行う場合, このようなシーンで意図した動きを得ることは大変困難である (場所により反射係数を変えるなどの工夫が必要である).

図 10 は, 現実には不可能なシミュレーションをもう一歩進めたものである. この例は, 我々のシステムのフレキシブルさを表しているものと言える.

図 11 はもう一つの例である. パーティクルは屋根に衝突したのちに, 右下のキャラクタの頭部 (図 2e,f で示したような球の表面状の法線が定義されている) に衝突するが, このような結果を返す三次元的なシーンを構築することはほぼ不可能である. なぜなら, 屋根では一旦衝突範囲が広がったのちに, 小さなキャラクタ頭部にパーティクルが当たらなければならないからで, それには奥行きまで考慮して屋根の法線を細かく調整し, そのような結果を返すような三次元の位置関係を探らなければならないからである. しかしながら, 我々のシステムでは容易にこのようなインタラクションを作りだすことができる.

5 まとめと今後の課題

我々のシステムは, 二次元画像の上に法線を描き, ユーザーが意図するシミュレーション結果を指定することにより簡単にパーティクルシミュレーションを行うことができる. これにより, シミュレーションの指定が簡単になったのみならず, 表現力もより増したということができる. このように, 法線を用いた二次元画像の利用価値が高まったと考えているが, 本システムにはまだ克服すべき課題がある.

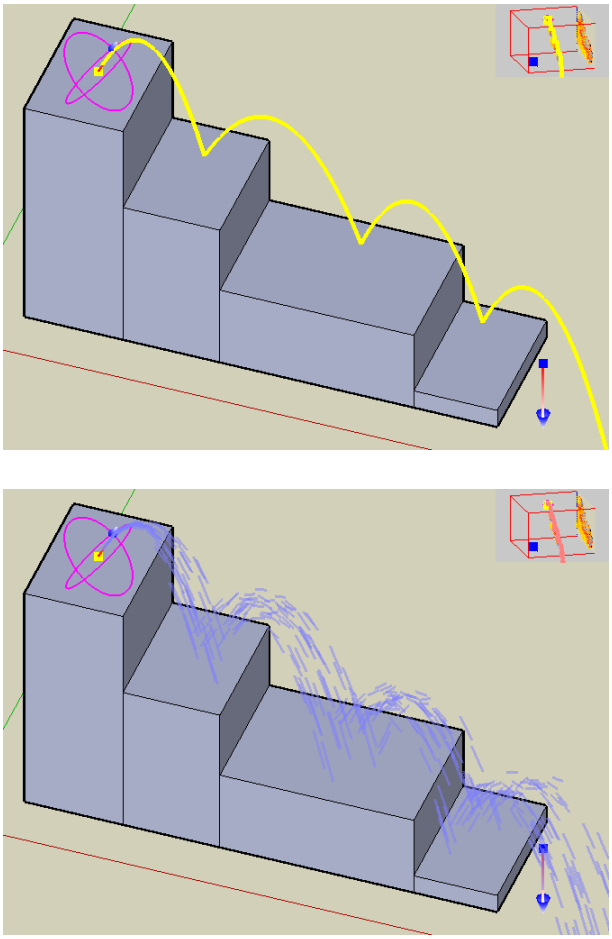


図 9. 結果 1

我々が最も大きな課題と考えているのは, 各衝突を全てユーザーが指定しなければならないという点である. そもそも CG におけるシミュレーションは, 物体の動きを手動で与えることが煩わしいために自動計算しているとも考えることができる. 我々は法線と初期条件を与えることにより, 空中での軌跡, および衝突処理を物理計算して自動化してはいるが, ユーザーが指定していない衝突は決して起こらないのである. この問題を解決するためには, システム自体にわたる多くの修正が必要であると思われる.

まず, この「指定していない (が, ユーザーが期待する) 衝突」を起こすためには, 局所的にであってもいいので三次元形状を再構築する必要がある. しかし, この法線からの三次元形状再構築問題は, 何度も述べているように, 原理的に不可能である. 我々は, ユーザーがガイドを与えることにより, 近似的に三次元形状を再構築する方法を研究している. また, この局所形状再構築を, どの程度の「局所」にわたって実行すべきかも問題である. これにもユーザーの指定が必要ではないかと我々は考えている. また, これに関連して, 連続的に接触している物体の動きを表現で

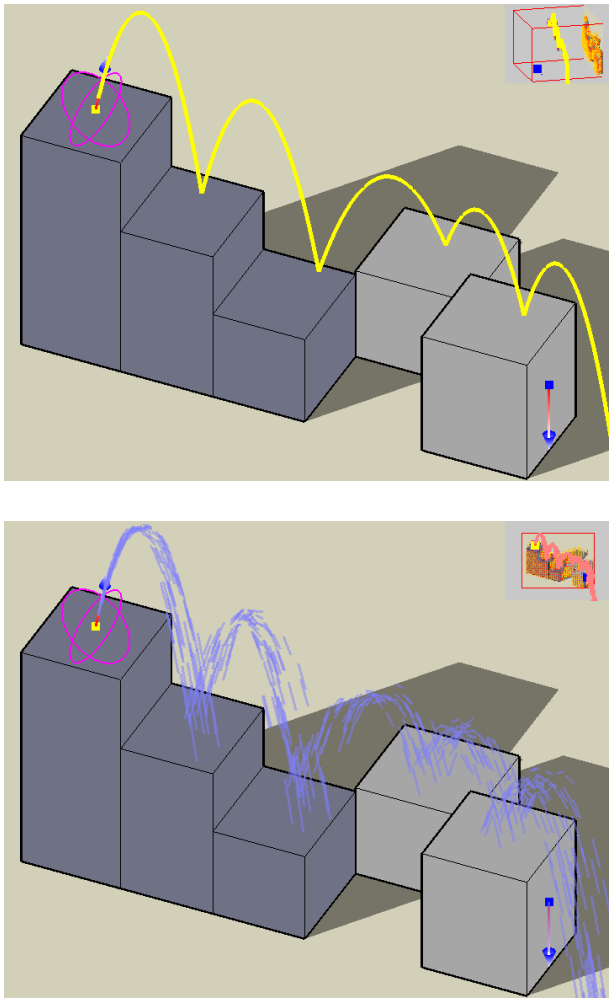


図 10. 結果 2

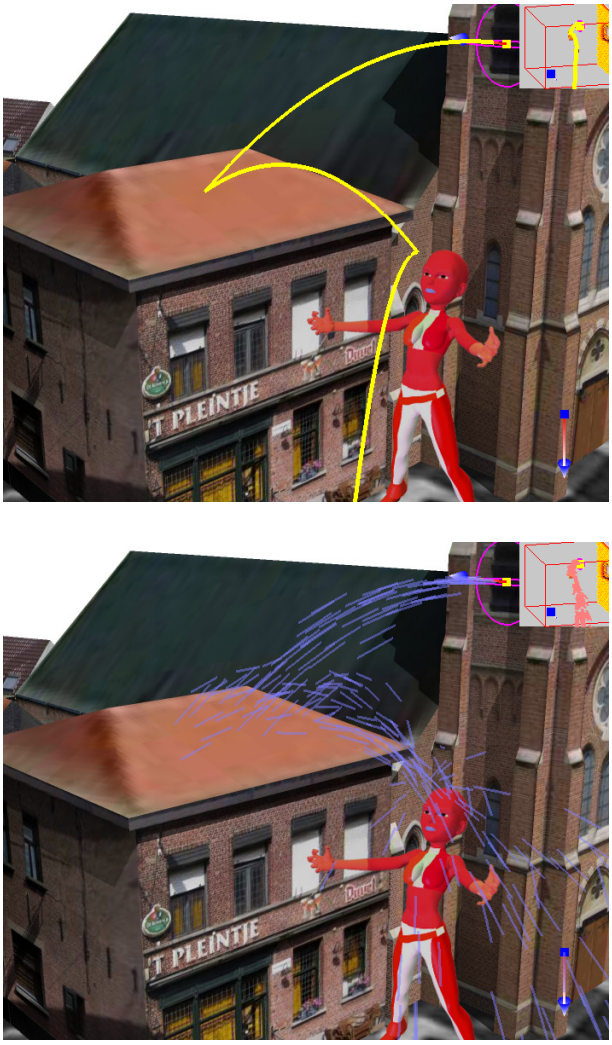


図 11. 結果 3

きない. 例えば, 坂道を転がっていく物体は離散的に衝突が起こらない. このような現象もシミュレーションできるように拡張を行いたいと考えている. さらに, パーティクル間に働くインタラクションの計算ができないことも欠点である. この問題も, 局所的な三次元構築の問題と関係がある. なぜなら, パーティクル間に (三次元的に働く) 力を計算するには, パーティクル間の相対的な位置関係が定義されている必要があるからである. このことは, 粒子法により流体を表現する場合にも問題であるし, より単純には, パーティクルをレンダリングするさいに, 遠近関係が定義されていないという問題にも反映されている. 現在では前後関係が曖昧なために, 各パーティクルは速度に応じた流さを持った線として描かれている (この場合は描画順が結果画像のクオリティに影響しない) が, 将来的に各パーティクルをメタボールで置き換えて流体らしい表面形状を得るためには前後関係が大変重要である.

次に, 一ピクセルあたり法線が一つしか定義できないため, 遮蔽のある物体を表現できない. これに関しては, 一ピクセルに複数の法線を割りあてたり, あるいは法線マップを遠近順に複数用意するなどして対処可能だと思われる.

また, 現在ではパーティクルの軌跡は正射影によって画面に表示されているが, 透視投影をサポートすることも必要である.

いずれにしても, 法線画像の可能性はまだ探求しつくされていないと考えられる. 本稿で提案した基本的なシミュレーション以外にも様々なインタラクションが可能であると思われるため, 今後様々に発展することが期待される.

参考文献

[1] J. Foley, A.van Dam, S. Feiner, and J. Hughes. Computer Graphics: Principles and Practice, second edi-

tion, 1990.

[2] A. Bernhardt, A. Pihuit, M. P. Cani, and L. Barthe. Matisse: Painting 2D regions for Modeling Free-Form Shapes. *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '08)*, pp.57-64, 2008.

[3] Y. Gingold and D. Zorin. Shading-based surface editing. *ACM Trans. Graph. (Proc. Siggraph '08)*, 2008.

[4] J. McCann and N. Pollard. Real-time gradient-domain painting. *ACM Trans. Graph. (Proc. Siggraph '08)*, 2008.

[5] M. Okabe, Y. Matsushita, L. Shen, and T. Igarashi. Illumination Brush: Interactive Design of All-Frequency Lighting. Proc. *Pacific Graphics '07*, pp.171-180, 2008.

[6] P. Pérez and M. Gangnet, and A. Blake. Poisson image editing. Proc. *ACM Trans. Graph. (Proc. Siggraph '03)*, pp.313-308, 2003.

[7] P. Joshi, and N. Carr. Repoussé: Automatic Inflation of 2D Artwork. *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM '08)*, pp.49-55, 2008.

[8] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa and C. Rössl, and H.-P. Seidel. Laplacian surface editing. Proc. *Symposium on Geometry Processing '04*, pp.175-184, 2008.

[9] R. Zhang, P. Tsai, J. Edwin Cryer and M. Shah. Shape from Shading: A survey. *Trans. PAMI* 21, pp.690–706, 1999.