パステルの質感を持つストロークのレンダリング技法に関する研究

村上, 恭子

https://doi.org/10.15017/458912

付録

A　ストローク生成のためのプログラム（C 言語）

A．1　構造体とグローバル変数

```
typedef struct tabStrokePrm{
        double x;          //x 座標
        double y;          //y 座標
        double ua;         //x 方向のストロークベクトル
        double ub;         //y 方向のストロークベクトル
}StrokePrm;

StrokePrm mainpoint[500];   // 基本直線を構成する点を格納する配列
double minlength ;          // ストロークの最小長さ
```

A．2　ストローク生成関数

```
// ストロークをバッファ上に描画する関数

void Stroke( double x1, double y1,          // 始点の座標値
             double x2, double y2,          // 終点の座標値
             double strokewidth,            // ストロークの幅
             double r, double g, double b,  // ストロークの色
             int type,                      // 描画手法のタイプ
             int planenum)                  // 面番号（オブジェクト描画時）
{
    int divnum;   // ストロークのセグメント数

    // 基本直線生成
    divnum = StrokeBasis(x1,y1,x2,y2,planenum);

    // 各セグメントにおけるストロークベクトルを計算
    for(int i=0; i<divnum; i++)
        CalcVector(mainpoint[i].x, mainpoint[i].y, mainpoint[i+1].x, mainpoint[i+1].y,i );
    // 最後のセグメントは始点・終点の座標から算出
    CalcVector(mainpoint[0].x, mainpoint[0].y, mainpoint[divnum].x, mainpoint[divnum].y,divnum);

    for(int j=0; j<divnum; j++){
        // 基本直線の各セグメントに補間ストローク生成
        InterpolationStroke(mainpoint[j].ua, mainpoint[j].ub, mainpoint[j].x, mainpoint[j].y,
                                                    minlength, strokewidth, r, g, b, j, type, planenum);
    }
}
```

A．3　基本直線生成関数

```
// 基本直線を生成し，セグメント数を返す関数

int StrokeBasis( double x1, double y1,          // 始点の座標値
                 double x2, double y2,          // 終点の座標値
                 int planenum)                  // 面番号（オブジェクト描画時）
{
```

```
    double length;
    double x,y;
    int divnum=0;           // ストロークのセグメント数（初期値を 0 にセット）
    double strokeblur=4.0;  // 基本直線のぶれ

    // 始点ー終点間の距離 length を測る
    length = sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));

    //minlength に最も近い値で length を割り切る
    while((length / (double)divnum) > minlength)
        divnum++;

    int *rnd_p2=&Rnd[planenum%RNDMAX]; // 面番号をキーとしてランダム数列を引き出し

    // 分割回数に従って分割位置を格納する
    for(int i=0; i<=divnum; i++){
        x = x1 + (x2 - x1) / divnum * i;
        y = y1 + (y2 - y1) / divnum * i;
        if(i>0 && i<divnum){
            mainpoint[i].x = x + (double)(*rnd_p2)/(RAND_MAX) * strokeblur-strokeblur/2;
            mainpoint[i].y = y + (double)(*rnd_p2++)/(RAND_MAX) * strokeblur-strokeblur/2;
        }else{
            mainpoint[i].y = y ;
            mainpoint[i].x = x ;
        }
    }
    return(divnum);
}
```

A．4　補間ストローク生成関数

```
// 補間ストロークを生成し，描画技法に従って顔料を付着させる関数

void InterpolationStroke(  float Vsx, float Vsy,          // ストロークベクトル
                           double Sx, double Sy,          // ストロークの始点
                           double Length,                 // セグメントの長さ
                           double strokewidth,            // ストロークの幅
                           double r, double g, double b,  // ストロークの色
                           int segmnum,                   // セグメント番号
                           int type,                      // 描画手法のタイプ
                           int planenum)                  // 面番号（オブジェクト描画時）
{
    int Bias2Width;
    int Xbias=10,Ybias=1;          // 幅のぶれにつける偏りの調節
    double Interval=0.1;           // 顔料を置く間隔 Δd
    double a,b;                    // 変化数
    double th1,th2;
    float SxNew,SyNew;             // ストロークの基本直線上の点

    float SxNew_1,SxNew_2,SyNew_1,SyNew_2;
    float Xl_1,Xl_2,Yl_1,Yl_2;     // ストローク上の点
    double random0,random1,random2; // ぶれ計算用
    double SetWidth;               // 実際のストロークの幅
    float Vax, Vay, Vbx, Vby;      // 基本直線の両側へ向かうベクトル

    int *rnd_p2=&Rnd[planenum%RNDMAX];

    UnitVec(&Vsx,&Vsy);

    for(a=0 ; a<=Length ; a+=Interval){
```

```
SxNew = Sx + Interval * Vsx;          SyNew = Sy + Interval * Vsy;
Sx = SxNew;       Sy = SyNew;

// 幅全体に対してぶれを適用
if(*rnd_p2==NULL)          rnd_p2-=(RNDMAX-1);
random0=(double)(*rnd_p2++)/(RND_MAX);
Bias2Width=(int)(random0*Xbias+Ybias);// 幅のぶれに偏りをつける
switch(Bias2Width){
case 1:
     Xbias=2; Ybias=1;      break;
case 2: case 3: case 4:
     Xbias=5;Ybias=1;       break;
case 5: case 6: case 7:
     Xbias=5; Ybias=4;      break;
case 8: case 9: case 10:
     Xbias=4; Ybias=7;      break;
default:       break;
}
SetWidth = strokewidth + strokewidth / (-Bias2Width);

Vax = Vsy;          Vbx =-Vsx;
Vay =-Vsy;          Vby = Vsx;
SxNew_1 = SxNew;          SyNew_1 = SyNew;
SxNew_2 = SxNew;          SyNew_2 = SyNew;

// 幅を作る直線のぶれ
for(b=0; b<=SetWidth/2; b+=Interval){
     if(*rnd_p2==NULL)     rnd_p2-=(RNDMAX-1);
     random1 =(double)(*rnd_p2++)/((RAND_MAX)-1.0);
     th1=random1*(-180)+90;
     random2 =(double)(*rnd_p2)/((RAND_MAX)-1.0);
     th2=random2*180-90;

     Vax =  Vsy + (-Vbx)* sin((double)th1 * rad)+ Vax * cos((double)th1 * rad);
     Vbx = -Vsx +  Vbx* cos((double)th1 * rad)+ Vax * sin((double)th1 * rad);
     Vay = -Vsy +(-Vby)* sin((double)th2 * rad)+ Vay * cos((double)th2 * rad);
     Vby =  Vsx +   Vby* cos((double)th2 * rad)+ Vay * sin((double)th2 * rad);

     UnitVec(&Vax,&Vbx);        UnitVec(&Vay,&Vby);

     Xl_1 = SxNew_1 + Vax * Interval;      Yl_1 = SyNew_1 + Vbx * Interval;
     Xl_2 = SxNew_2 + Vay * Interval;      Yl_2 = SyNew_2 + Vby * Interval;
     SxNew_1 = Xl_1;          SyNew_1 = Yl_1;
     SxNew_2 = Xl_2;          SyNew_2 = Yl_2;

     /ストロークバッファに顔料を保存
     if(/* 顔料付着位置がスクリーンの内部なら */){
          switch (type,){
          //エフェクトなし
          case 0:    NoneStroke(Xl_1, Yl_1, Xl_2, Yl_2, segmnum, r, g, b, planenum);    break;
          //   一般的なストローク
          case 1:    NormalStroke(Xl_1, Yl_1, Xl_2, Yl_2, segmnum, r, g, b, planenum); break;
          //サイドストローク技法
          case 2:    SideStroke(Xl_1, Yl_1, Xl_2, Yl_2, segmnum, r, g, b, planenum);    break;
     }
   }
  }
}
```

B　パステル風アニメーション生成のための流れ図

アニメーションに利用するモデル及びシーンは LightWave7.0 で作成し，D Storm (http://www.dstorm.co.jp/) より提供されているライブラリを用いてプログラム内で利用している．

B．1　main 関数

全体の流れを統率する関数．C 言語では常にこの関数から実行される．

① 初期化処理

以下の３つの関数群からなる関数．

1．ランダムテーブル作成

rand 関数を用いてランダムな数列を発生させ，テーブルに確保する．

2．紙面の定義

紙面となる Height Field（ビットマップファイル）を読み込み，スクリーン上に敷き詰める．

3．Z バッファのクリア

Z バッファに無限遠（INFINITY=le5）を格納．

② アイテム読み込み

LightWave シーンを開き，シーン中のアイテムを読み込む処理．

③ アイテムの種類

LightWave シーン中のアイテムの中で，照明やカメラといったアイテムを取り除き，ポリゴンで構成されるオブジェクトアイテムのみを取り出す分岐処理．

④ NULL オブジェクト判定

オブジェクトアイテムに分類される，実体を持たないオブジェクト（NULL オブジェクト）を取り除く分岐処理．

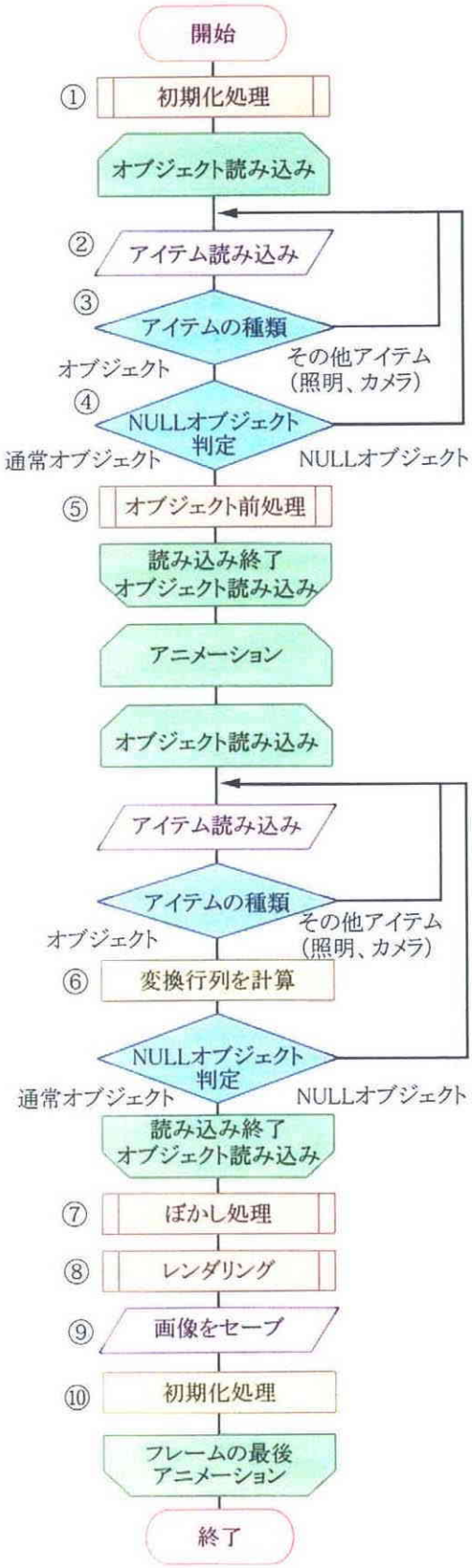


図 B － 1．main 関数の流れ図．

⑤ オブジェクト前処理

PrepareObj 関数（B. 2) にて解説.

⑥ 変換行列を計算

各フレームでオブジェクトにかかる変換行列を計算する処理.

⑦ レンダリング

Rendering 関数 (B. 5) にて解説.

⑧ ぼかし処理

紙面上に付着した顔料をガウス関数を用いた処理によってぼかす処理を行う関数.

⑨ 画像をセーブ

ストロークバッファに蓄えられた顔料を，背景紙面を考慮しつつ画像としてセーブする関数.

⑩ 初期化処理

Z バッファ及びストロークバッファの初期化.

B. 2 PrepareObj 関数

オブジェクト上にパーティクルを発生させ，パーティクルを始点として面上にストロークエレメントを配置する関数. パーティクル及びストロークエレメントはストローク位置を決定する.

① *poly_num*

オブジェクトを構成するポリゴンの数.

② 面の接続関係を構築

四角ポリゴンの四辺に接続するポリゴンの番号を辺上に保存.

③ 面積比計算

最小面積のポリゴンを基準とした，全ての面の面積比を計算.

④ 特徴線探索

接続する面の角度によって，その間の辺を特徴線とするかどうかを決定.

⑤ パーティクル散布

SetParticle 関数（B. 3）にて解説.

⑥ ストロークエレメント設置

CalcParticle 関数（B. 4）にて解説.

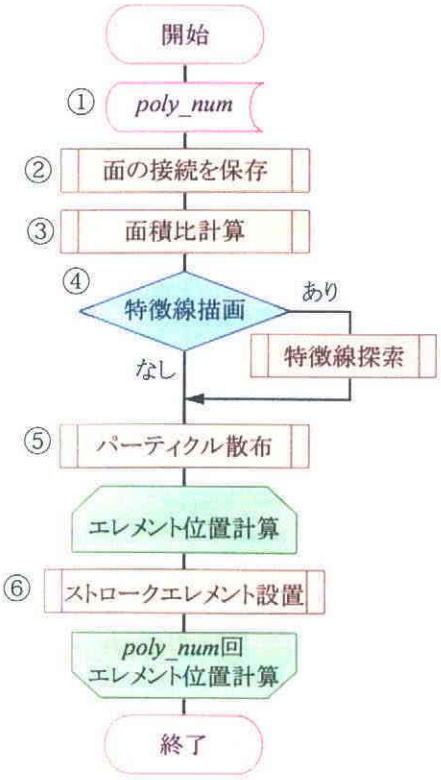


図 B － 2. PrepareObj 関数の流れ図.

B．3　SetParticle 関数

① サーフェスデータオープン
②－⑦のパラメータを記述したデータを開く．これらのデータはユーザが入力する．

② スパイラル
図３－12の球体や，図３－13の葡萄の実のような，らせん状（スパイラル）のストロークを描画する際には，スパイラル開始ポリゴンを設定する．

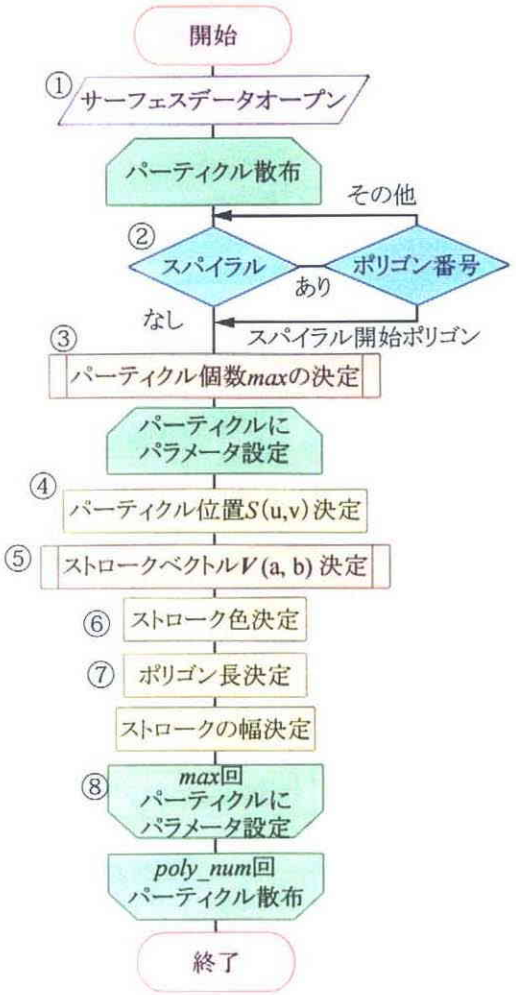
③ パーティクル個数 *max* の決定
図Ｂ－２③で計算した面積比を用いるか，①で設定する．

④ パーティクル位置 *S(u,v)* 決定
ランダムを用いるか，①で設定する．

⑤ ストロークベクトル ***V(a,b)*** 決定
①では基本的な値のみを決定し，技法によって図Ｂ－４のように変化させる．図中 *j* は 0 ～ *max* までの数．

⑥ ストローク色決定
LightWave 上で与えた色を読み込むか，①で決定する．

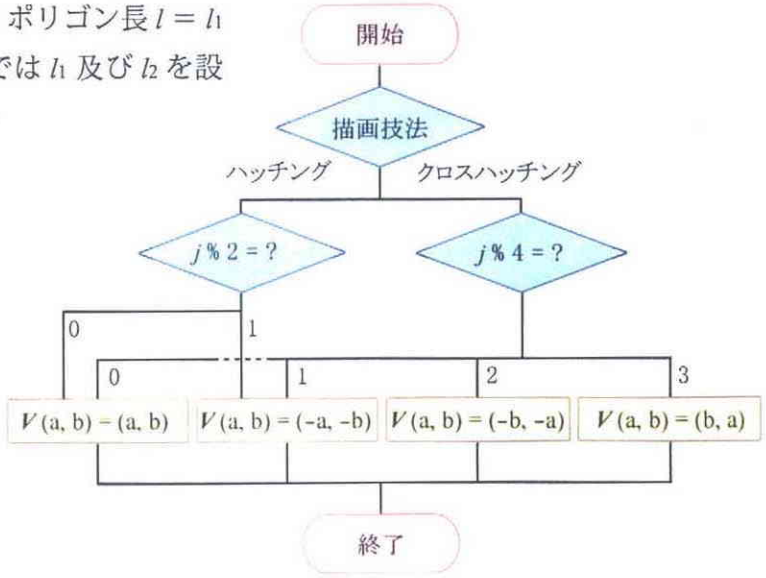
⑦ ポリゴン長決定
長さに変化をつけるため、ポリゴン長 $l = l_1 + \mathrm{rand}[0\text{–}1]^* l_2$ とする．①では l_1 及び l_2 を設定する．



図Ｂ－３．SetParticle 関数の流れ図．

B．4　CalcParticle 関数

① 初期パーティクル数 *n*
SetParticle 関数において散布したパーティクルの数．



図Ｂ－４．技法の決定．

② パーティクルの値読み出し
SetParticle 関数においてパーティクルに与えられたデータを読み込む.

③ 始点前境界点 *B(Bu, Bv)* の算出
始点前境界点を計算し，終点 *E* の後に付け加えるポリゴン長 *pl* を算出する.

④ ループ回数 *n* の決定
設定されたポリゴン長 *l* と付加されたポリゴン長 *pl* から，実際に渡るポリゴンの数を決定する.

⑤－⑧ 終点の判定
ポリゴン長に小数部がある場合の処理．ポリゴン上での値によって終点となるか，もう 1 ポリゴン進む.

⑨⑩ 境界点の処理.
現在のポリゴン上に境界点 *en* のパラメータを持つパーティクルを保存.

⑪ 接するポリゴン
境界点を共有するポリゴンを検索.

⑫⑬ 接するポリゴンへの値渡し
接するポリゴン上に境界点をパラメータと共に渡し，現在処理しているポリゴンを接するポリゴンへと切り替える.

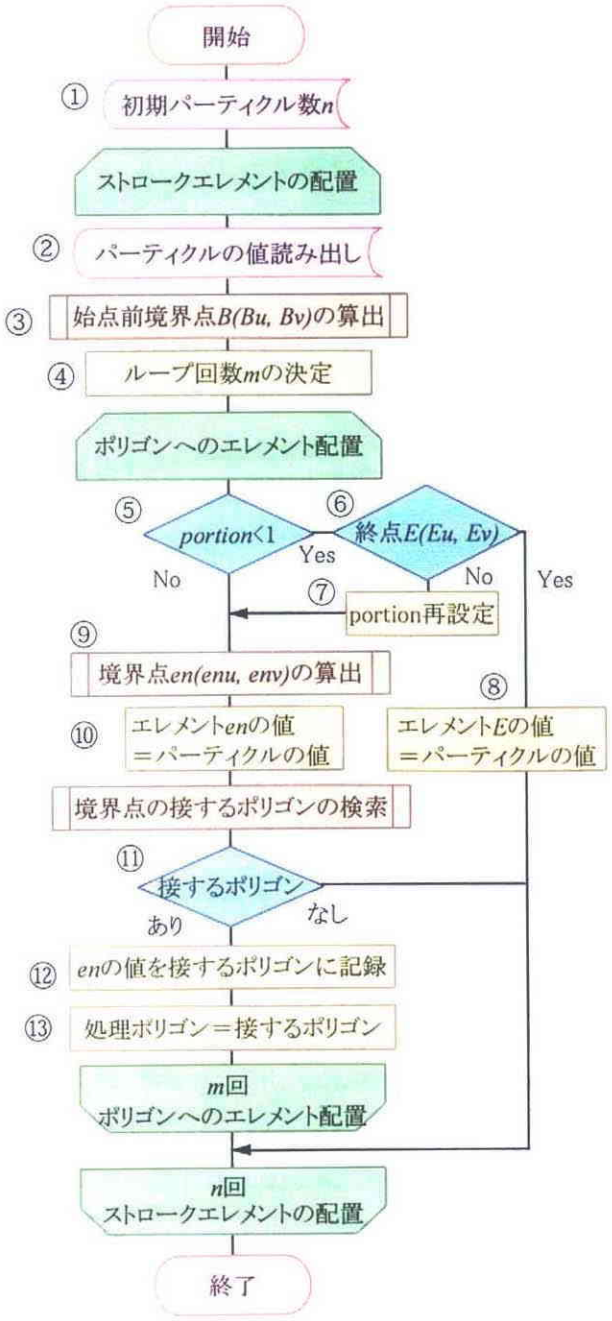


図 B － 5. CalcParticles 関数の流れ図.

B. 5 Rendering 関数

① オブジェクトデータの読み込み
頂点数 *point_num*, ポリゴン数 *poly_num*, ストロークエレメントを含むデータを読み込む.

② アフィン変換
頂点座標に対し，LightWave 上での設定に従って移動・回転・拡大縮小等の変換処理を行う.

③ 透視投影
透視投影を行って二次元スクリーン上に頂点座標を投影.

④ デプステスト

隠面消去のための準備．前方面を判定し，スクリーンから遠い順に面をソートする．

⑤ ストローク描画

面上に保存されたパーティクルを始点及び終点として，付録Aのストローク描画関数を用いて描画を行う．このとき顔料は，全体の顔料が保存されているストロークバッファではなく，現在描画中のポリゴン用バッファに一時的に保存される．同時に，現在のポリゴン用のZバッファも生成される．

⑥ Zバッファ作成

これまでのZバッファと現在のポリゴン用のZバッファを比較して前後判定を行い，新たなZバッファを作成する．同時に，後方にあると判定された部分のポリゴン用バッファあるいはストロークバッファZバッファから顔料を消去し，足し合わせて新たなストロークバッファを得る．

図B－6．Rendering関数の流れ図．

C　ドローイングツール生成のための流れ図及びプログラム

筆圧取得の可能なドローイングツール生成のため，WintabSDK（http://www.pointing.com/）を用いてタブレットデバイスを利用したプログラムを行っている．ここでは，筆圧データを含むポインティング位置データが入力された場合に行われる処理について詳細を記す．

C．1　流れ図

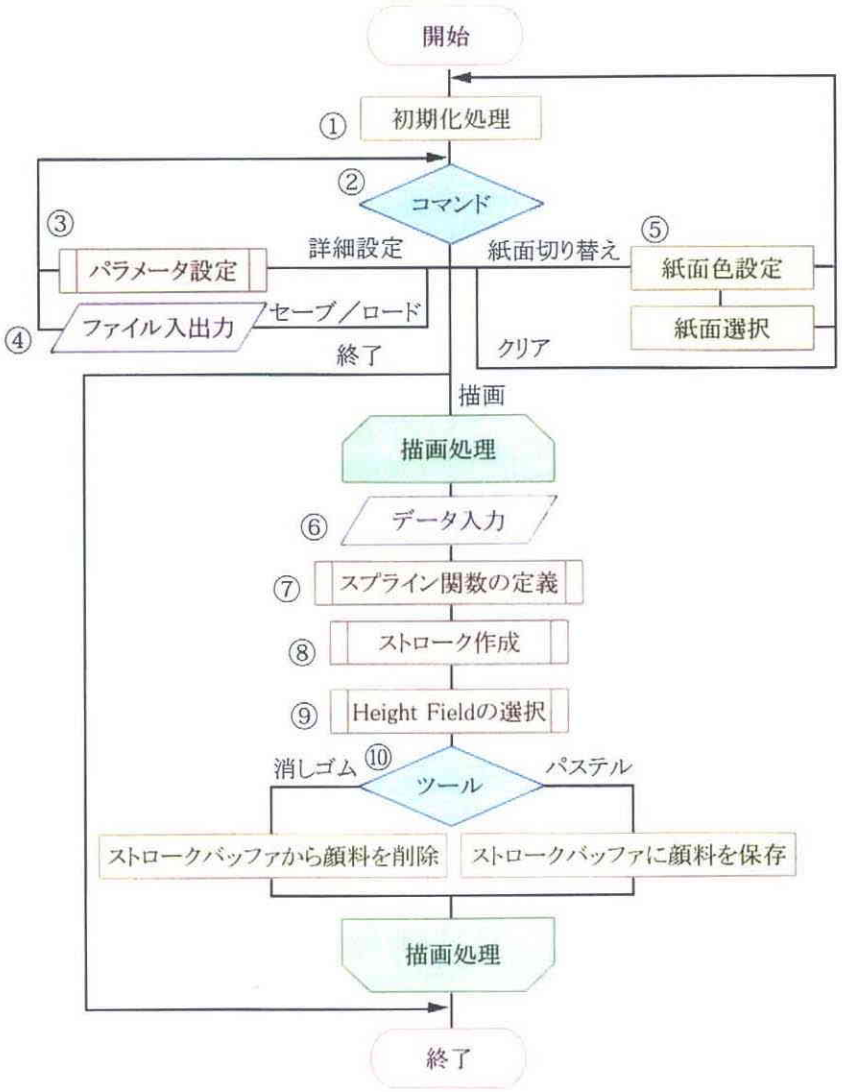
図C－1はドローイングアプリケーションの流れ図を示している．アプリケーションが実行されると，初期化の後，WindowProc関数によりユーザの入力を取得して逐次処理を行っていく．

① 初期化処理

紙面を読み込み，コントラスト強調補正を行う．アプリケーションで利用するパラメータを初期化する．

② コマンド

メニューコマンド（セーブ／ロード，パステル／消しゴムツール切り替え，パラメータ設定，紙面切り替え，画面クリア）及びクライアント領域に描画することによって分岐



図C－1．ドローイングアプリケーションの流れ図．

する．

③パラメータ設定

ストローク描画に用いるパラメータを設定する．パラメータは筆圧補間方式（線形／二次式，二次式の場合は係数），ストローク色，ストロークの幅．

④ ファイル入出力

描画したファイルを開くあるいは保存する．

⑤ 紙面切り替え

紙面の色を選択．また，紙面の種類を画用紙・コットン紙・ワトソン紙・マーメイド紙・キャンソン紙の５種類から選択．

⑥ データ入力

タブレットペンを用いてx，y座標と筆圧を入力する．

⑦ スプライン関数の定義

入力データである点列を補間し，基準となるスプラインを作成．補間には全ての点を通

過するスプライン関数である Catmull-Rom Spline を用いる．

⑧ ストローク作成

スプラインに幅をつけ，筆圧を線形または二次式で補間してストローク上に閾値を設定する．

⑨ Height Field の選択

ストロークベクトル及び圧力ベクトルから Height Field を選択する．詳細については C. 2 のプログラムにおいて記す．

⑩ ツール

パステル：

⑧で設定した閾値と⑨で得た Height Field 値により顔料の付着量を決定し，ストロークバッファ上に顔料を保存する．

消しゴム：

⑧及び⑨で得た値より顔料の削除量を決定し，ストロークバッファから顔料を削除する．

C．2　Height Field 選択プログラム

```
// ベクトルからテクスチャを選択する関数

void TextureSelect( float vecx, float vecy,          // ベクトル
                    int *k1, int *k2,                // 選択されたテクスチャ
                    float *asp)                      // ベクトルの角度の比
{
    float th;       // ベクトルと x 軸のなす角度
    float s;        // 照明方向ベクトルと x 軸のなす角度
    int angle = 30;// 照明方向ベクトルの角度のインターバル
    int j=0;
    int min, max;
    // ベクトルの方向合わせ
    th = (float)acos(vecx) / 3.141592f * 180;
    if(vecy<0)   th = 360 - th;

    s = 360-th-90;
    if(s<0)   s+=360;

    // 照明方向ベクトルの検索
    for(int i=0; i<360; i+=angle){
        min = i;   max = i+angle;
        if(s>=min && s<max){
            *k1 = j;
            if(max!=360)    *k2 = j+1;
            else *k2=0;
                *asp = ((i+angle)-s)/angle;
        }
        j++;
    }
}
```