松 山 大 学 論 集
第 29 巻 第 1 号 抜 刷
2017 年 4 月 発 行

WebGL による線香花火の CG シミュレーション

檀　　裕　也

WebGL による線香花火の CG シミュレーション

檀　　裕　也

1　は　じ　め　に

コンピュータグラフィックス（CG）の映像表現に写実性を実現するため，数学および物理を用いた計算モデルが使われるようになってきた。ニュートンの運動方程式に基づく力学および剛体力学は，インタラクティブな映像表現における落体の法則や衝突判定に応用されていることは周知の事実である。また，流体力学を用いると，水の流れや空気の流れはもちろん，炎や煙といった自然現象の写実的な映像表現を可能にする大きな効果がある。さらに，ハミルトン系を中心に記述する解析力学は，その数学的な構造を拡張することによって量子力学に昇華させることになった。もちろん，量子力学の基礎方程式は，量子状態を波動関数として記述するシュレーディンガー方程式である。2014年にガウス賞を受賞したオッシャー［1］は，ハミルトン・ヤコビの偏微分方程式を用いて特定の画像領域の時間変化をトラッキングする「レベルセット法」を提案した。従来は，粒子法（パーティクル法）やシミュレーションに基づく計算で写実的な映像表現をしていたところに，計算コストを大幅に削減したレベルセット法が導入されたことによって，（画像処理用に特化したワークステーションではなく）比較的スペックの低い通常のコンピュータであっても実時間でレンダリングすることができるようになった。

現在，写実的な映像表現としては，例えば「アナと雪の女王」のような映画作品で流体や結晶の表現手法が商業ベースで実用化されている段階である。さらに，計算コストの制約の中で，新しく効果的な映像表現手法について提案で

きることが予想される。すでに，偏微分方程式の研究は数学研究の一部門として十分な知見が得られている。一方，工学的および芸術的なアプローチからは映像表現のレンダリングに関する研究が進展している。本研究課題は，両者の組み合わせによる新しい領域を切り開こうとするもので，国際的には数年のうちに競争分野として発展が見込まれる状況にある。

そこで，本研究では写実的な表現として自然現象をシミュレーションできるプログラムを開発した。物理法則に基づく自然現象の可視化，数学・物理表現による映像表現その他のCGについて，ブラウザベースで動作するHTML5のcanvas＋JavaScript［2, 3］で実装するほか，WebGL［4］によるコンテンツを制作し，その性能を評価する。

なお，本研究は2015年度に交付を受けた松山大学特別研究助成による成果の一部である。

2　物理法則の数学的表現

ニュートン力学に基づく質点の運動を考えよう。いま質量 m の点が外力 $F(t,\ x)$ を受けている物理系を設定する。外力はベクトルであって，一般に時刻 $t\in\mathbb{R}$ および位置 $x\in\mathbb{R}^3$ の関数として与えられる。すると，時刻 $t\in\mathbb{R}$ における質点の位置 $x(t)$ は，微分方程式

$$m\frac{d^2}{dt^2}x(t)=F(t,\ x)$$

によって表すことができる。これがニュートンの運動方程式である。

外力 $F(t,\ x)$ が与えられたときに，質点の運動，すなわち $x(t)$ を求めるには時刻 $t\in\mathbb{R}$ に関する積分を実行すればよいが，コンピュータでは無限小を扱えないため，微小時間 Δt に関する差分方程式を考え，近似的に微分方程式を解く。外力が重力しかない場合

$$F(t,\ x)=-g^t(0,\ 1,\ 0)$$

と置けばよい。また，外力が空気抵抗を含むような場合には，

$$F(t,\ x) = -k\frac{d}{dt}x(t)$$

を加えればよい。ただし，$k > 0$ は空気抵抗の大きさを表す係数である。質点が複数あるような物理系では，ニュートンの運動方程式を連立して解けばよい。与えられる外力が上記のように適切な場合には，解の存在と一意性が保証されている。

3　HTML5 Canvas 2D Context による CG 表現

2016 年 11 月 1 日に World Wide Web Consortium（W3C）によって勧告された HTML 5.1 には，引き続き canvas 要素として，２次元平面座標におけるグラフィックスの描画処理を JavaScript で実装できる Canvas 2D Context が備わっている。標準 HTML 形式のテキストファイルとして必要な API を呼び出すことで，簡単にグラフィックスの描画や画像ファイルの読み込み等が実行できるほか，JavaScript の setTimeout 関数などと組み合わせることによってアニメーションを実現することができる。

Canvas 2D Context の機能によって表示したい Web ページ上の場所に HTML の canvas 要素を加える。その際，JavaScript 側から呼び出すときの id 属性を指定するほか，必要に応じて canvas 要素のサイズを与える。JavaScript のソースコードは HTML の script 要素の中に記述する。エントリポイントとして HTML 中の body 要素に onload イベントハンドラを定義しておき，Web 画面の描画時に drawScreen 関数を呼び出す。グラフィックスの描画処理は Canvas 2D Context を通じて各種の API を実行することになる。また，アニメーション処理を実装するには setTimeout 関数を用いて適宜描画処理を繰り返し呼び出せば良い。１秒あたりの描画フレームについては，30〜60［fps］程度あれば十分である。そのため，setTimeout 関数を呼び出すタイミングは 33 ミリ秒または 16 ミリ秒にすることが多い。

Canvas 2D Context を用いて CG 表現を実装する HTML ファイルの基本的な構造は，以下の通りである。

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>タイトル部</title>
<script>
// 初期設定
function drawScreen(){
    var canvas = document.getElementById( "myCanvas" );
    var myContext = canvas.getContext( "2d" );

    // 計算処理

    // 描画処理

    setTimeout( 'drawScreen()',33 );
}
</script>
</head>
<body onload="drawScreen();">
<canvas id="myCanvas" width="640" height="480">
    canvas 非対応時の処理
</canvas>
</body>
</html>
```

3.1 ホタルの乱舞

自然現象のシミュレーションとして，ホタルの乱舞を CG によって表現する。初期設定，計算処理および描画処理の順で述べる。

(1) 初期設定

まず，パーティクル法を用いて 16 匹のホタルの位置および速度を表すために，それぞれ（**x**，**y**）および（**vx**，**vy**）の配列をメモリ上に用意する。簡単のため，ホタルの位置は 3 次元空間座標から **x-y** 平面上への正射影とすることで最初から計算量を削減しても表現の一般性を失わない。また，空間のトポロジー構造にはトーラスを導入した。なお，初期値は乱数を用いて適切な範囲に収まるように工夫している。ここで，**Math.random** は 0 以上 1 未満の値を取る乱数メソッドである。単位は，位置はピクセル，速度は **30px/s** である。

ホタルの発光状態は，周期的な明滅を表現するため，0〜2π の値を取る位相（**theta**）を用いて計算する。

初期設定のソースコード（抜粋）

```
var x  = new Array(); //ホタルの x 座標
var y  = new Array(); //ホタルの y 座標
var vx = new Array(); //ホタルの x 方向の速度(px/33ms = 30px/s)
var vy = new Array(); //ホタルの y 方向の速度(px/33ms = 30px/s)
var theta = new Array(); //ホタルの発光状態の位相
for( var j = 0; j < 16; j++ ){
    x[j]  = Math.random() * 640;
    y[j]  = Math.random() * 480;
    vx[j] = Math.random() * 3-1.5;
    vy[j] = Math.random() * 0.5-0.25;
    theta[j] = Math.random() * 2 * Math.PI;
}
```

(2) 計算処理

次に，ホタルの位置について時間変化を計算するために，ニュートンの運動方程式を用いる。ホタルの運動意思はランダムであると仮定して，水平方向および垂直方向の加速度として，それぞれ

```
( Math.random() * 2 - 1 ) * 0.2
```

および

```
( Math.random() * 2 - 1 ) * 0.1
```

を与えた。このような加速度の仮定は，勝手に与えたものであるが，完成したホタルの動きを観察すると自然に見えるため，十分良い近似を与えていると考えられる。

空間のトーラス構造は，画面幅による剰余演算子を用いてモジュロ計算を実行している。

計算処理のソースコード（抜粋）

```
for( var i = 0; i < 16; i++){
    vx[i] += ( Math.random() * 2 - 1 ) * 0.2;
    if( vx[i] < -3 || 3 < vx[i] ){
        vx[i] /= 2;
    }

    vy[i] += ( Math.random() * 2 - 1 ) * 0.1;
    if( vy[i] < -0.5 || 0.5 < vy[i] ){
        vy[i] /= 2;
    }

    x[i] = ( x[i] + vx[i] + 640 ) % 640;
    y[i] = ( y[i] + vy[i] + 480 ) % 480;
}
```

(3) 描画処理

各時刻において各ホタルの位置（**x**, **y**）に円の描画メソッド **arc** を用いて中心から外側に向かってグラデーションで塗りつぶした。その際，RGB 値で設定したホタルの基本色（**128,255,0**）に関する情報［5］に加え，アルファブレンド値 **a** によって不透明度を与えて表現した。位相は毎回のフレーム更新時に **0.05** だけ変化させ，三角関数によってホタルの光の明滅を表現した。

描画処理のソースコード（抜粋）

```
myContext.clearRect( 0, 0, 640, 480 );
for( var i = 0; i < 16; i++ ){
    var a = Math.floor( ( Math.sin( theta[i] ) + 1 ) * 256) /256;
    theta[i] += 0.05;

    var g
    = myContext.createRadialGradient( x[i], y[i], 0, x[i], y[i], 16 );
    g.addColorStop(  0/10, "rgba( 128, 255, 0, " + a + " ) " );
    g.addColorStop(  1/10, "rgba( 128, 255, 0, " + a + " ) " );
    g.addColorStop( 10/10, "rgba(   0,   0, 0, " + 0 + " ) " );
    myContext.fillStyle = g;
    myContext.beginPath();
    myContext.arc( x[i], y[i], 16, 0, 2 * Math.PI, true);
    myContext.fill();
}
```

以上のアルゴリズムに従って表現したホタルの乱舞は**図 1** の通りである。また，ホタルの飛跡を**図 2** に示した。

図 1 ホタルの乱舞
http://www.cc.matsuyama-u.ac.jp/~dan/CG/work05.html

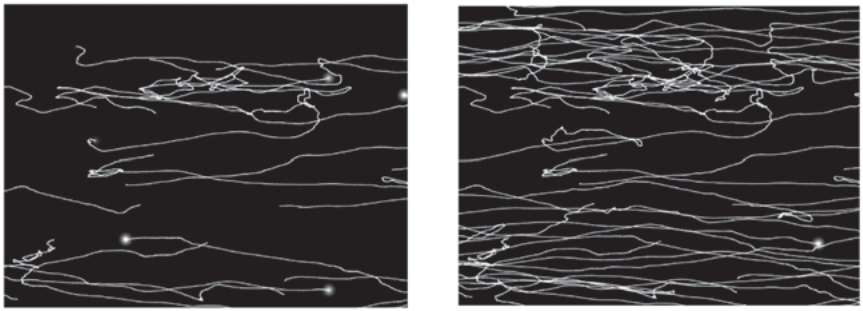

図 2 ホタルの飛跡

3.2 打ち上げ花火

パーティクル法の典型的な事例として，打ち上げ花火のシミュレーション CG を取り上げる。同様に，初期設定，計算処理および描画処理の順で述べる。

初期設定

花火のパーティクル 512 個は，位置および速度について，それぞれ（**x**，**y**）および（**vx**，**vy**）の配列で表すとともに，発光色を **color** の配列によって表すことにする。画面上のランダムな位置に出現する花火の発火点（**cx**，**cy**）を基準に，各パーティクルの速度（**vx**，**vy**）は球対称の確率分布によって与えることにする。すると，花火中心点から遠方に飛び出していく様子を表現することができる。

初期設定のソースコード（抜粋）

```
var x     = new Array(); //花火パーティクルの x 座標
var y     = new Array(); //花火パーティクルの y 座標
var vx    = new Array(); //花火パーティクルの x 方向の速度
var vy    = new Array(); //花火パーティクルの y 方向の速度
var color = new Array(); //花火パーティクルの色
var cx    = Math.random() * 512 + 256; //花火中心点の x 座標
var cy    = Math.random() * 256; //花火中心点の y 座標
for( var j = 0; j < 512; j++){
    x[j] = cx;
    y[j] = cy;
    var speed = 4 * Math.random();
    var theta = Math.random() * 2 * Math.PI;
    vx[j] = speed * Math.cos( theta );
    vy[j] = speed * Math.sin( theta );
    var red  = Math.floor( Math.random() * 128 ) + 128;
    color[j] = "rgb( " + red + ", 128, 128 )";
}
```

計算処理

花火のパーティクルにかかる力は重力のみである。したがって，加速度には重力加速度を与え，落下の様子を表現することができる。また，空気抵抗などに伴うブラウン運動を入れることも可能である。

計算処理のソースコード（抜粋）

```
for( i = 0; i < 512; i++ ){
    //vx[i] += 0.05 * ( Math.random() * 2 - 1 );
    vy[i] += 0.05;
    x[i] += vx[i];
    y[i] += vy[i];
}
```

描画処理

花火パーティクルの残像効果として，アルファ値 **0.05** によるブレンディングを入れ，飛跡のアニメーションを実現した。一つのパーティクルは 2 点間（**x**, **y**）および（**x+vx**, **y+vy**）を結ぶ太さ **4** の線分として表現している。

描画処理のソースコード（抜粋）

```
myContext.fillStyle = 'rgba( 0, 0, 0, 0.05 )';
myContext.fillRect( 0, 0, 1024, 768 );

myContext.strokeWidth = 4;
for( i = 0; i < 512; i++ ){
    myContext.beginPath();
    myContext.moveTo( x[i], y[i] );
    myContext.lineTo( x[i] + vx[i], y[i] + vy[i] );
    myContext.strokeStyle = color[i];
    myContext.stroke();
}
```

図 3　打ち上げ花火
http://www.cc.matsuyama-u.ac.jp/~dan/CG/work04.html

以上のアルゴリズムに従って表現した打ち上げ花火は**図 3** の通りである。

4　WebGL による線香花火の 3 次元 CG 表現

2016 年に開催された CG に関するトップレベルの国際会議 SIGGRAPH において，Chern らの研究グループがシュレーディンガーの煙という表現手法［6］を発表した。シュレーディンガー方程式とは，主にミクロな量子状態を記述する偏微分方程式であって，その解は複素数の値を取る波動関数である。非圧縮流体として表現されたシュレーディンガー流の物理的な意味は明らかではないものの，芸術的な表現手法としてメディアアートの分野では有力なものだと考えられる。

また，2017 年には Inoue らの研究グループが高性能カメラを用いて線香花火の現象［7］について明らかにした。実は，なぜ線香花火の火花が連続的に分岐して松葉模様を描くのかについては科学的に明らかではなかった。黒色火薬の化学成分および化学反応は知られていたものの，高温流体現象としての物理的解明が待たれていたというわけである。寺田寅彦や中谷宇吉郎といった日本人科学者たちを悩ませてきた問題［8］は，宇宙工学の研究で使われる毎秒 10 万コマの撮影が可能な高速度カメラによって世界で初めて観測されたものである。

そこで，発見された線香花火の分岐現象を 3 次元 CG 表現によって試行した。多数の粒子が 3 次元空間に広がる様子をリアルタイムに調べることができるように，WebGL によって実装した。

4.1 線香花火の原理

線香花火の主成分である硝酸カリウム（KNO_3）・硫黄（S）・炭素（C）の混合物である黒色火薬は，紙縒りの先端で

$$C + O_2 \rightarrow CO_2$$

という空気中の酸素との燃焼反応によって，10^7 J/kg の熱を発生することが知られている。実際，硝酸化合物は

$$4KNO_3 + 7C + S \rightarrow K_2CO_3 + K_2S + 3CO_2 + 3CO + 2N_2$$
$$2K_2CO_3 + (2x + 1)S \rightarrow 2K_2S_x + SO_2 + 2CO_2$$
$$8K_2CO_3 + (3x + 1)S_2 \rightarrow 6K_2S_x + 2K_2SO_4 + 8CO_2$$

といった化学反応を経て，K_2S（融点 1, 108 − 1, 221 K），K_2CO_3（融点 1, 164 K），K_2SO_4（融点 1, 342 K）による高温液滴が生じると同時に，高温液滴の内に CO_2 などの気泡が発生する。化学反応の進行によって高温液滴中の気泡が成長し，揺らぎの中で弾ける瞬間が液滴の分裂である。母液滴から飛び散った子液滴

は，さらなる化学反応によって孫液滴へと最大 8 回の分裂を繰り返す。その結果として，線香花火は美しい松葉模様の発光を見せるのである。

4.2 高温液滴内の気泡発生

高温液滴中の気泡の発生場所は，一様な確率で分布すると考えられる。そこで，高温液滴を半径 r の球であると仮定し，液滴の弾ける場所の座標を（x，y，z）で表すとき，極座標（θ，ϕ）を用いて次のような関係を満たす。

$$\begin{cases} x = r \sin\theta \sin\phi \\ y = r \sin\theta \cos\phi \\ z = r \cos\theta \end{cases}$$

ただし，$\theta \in [0, \pi]$ および $\phi \in [0, 2\pi)$ である。ところが，θ および ϕ の値を一様乱数によって与えることにすると，ヤコビアンの分だけ偏りが生じる。気泡の発生場所を一様とするため，棄却法を採用した。

棄却法による一様分布生成のソースコード（抜粋）

```
var x, y, z;
do{
   x = 2.0 * Math.random() - 1.0;
   y = 2.0 * Math.random() - 1.0;
   z = 2.0 * Math.random() - 1.0;
}while( x * x + y * y + z * z > 1.0 );
```

さらに，棄却法による気泡の発生場所の分布について，メルセンヌ・ツイスタ法および JavaScript の標準ライブラリに実装されている **Math.random** 関数の目視による比較によって差異がないことが確認されたため，以下の実装では JavaScript の標準ライブラリによる乱数生成を採用することにした（**図 4**）。

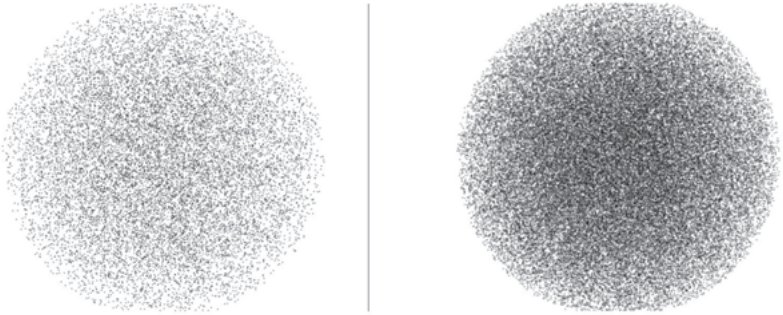

図 4　乱数 Math. random()の極座標一様分布

4. 3　高温液滴の連鎖的分裂

観測で得られた分裂のパラメータを用いてシミュレーションに実装した。液滴の分裂の前後では，エネルギー保存の法則および運動量保存の法則を満足するため，同一質量の子液滴に等分されるという仮定を入れた。

子液滴への連鎖的分裂のソースコード（抜粋）

```
var material = new THREE.SpriteCanvasMaterial({
    color: 0xffffff,
    program: program
});

var n = droplet.length;
var m;

do{
    m = Math.floor( n * Math.random() );
}while( droplet[m].visible == false );

droplet.push( new Droplet( material ) );
```

```
droplet[n].position.copy( droplet[m].position );

droplet[m].scale.x /= Math.cbrt(2.0);
droplet[m].scale.y /= Math.cbrt(2.0);
droplet[n].scale.x = droplet[m].scale.x;
droplet[n].scale.y = droplet[m].scale.y;

droplet[m].lifetime /=2;
droplet[n].lifetime = droplet[m].lifetime;

var s = PI  * Math.random();
var t = PI2 * Math.random();
var vx = droplet[n].scale.x * Math.sin( s ) * Math.sin( t );
var vy = droplet[n].scale.x * Math.sin( s ) * Math.cos( t );
var vz = droplet[n].scale.x * Math.cos( s );

droplet[n].velocity.x = droplet[m].velocity.x + vx;
droplet[n].velocity.y = droplet[m].velocity.y + vy;
droplet[n].velocity.z = droplet[m].velocity.z + vz;

droplet[m].velocity.x = droplet[m].velocity.x - vx;
droplet[m].velocity.y = droplet[m].velocity.y - vy;
droplet[m].velocity.z = droplet[m].velocity.z - vz;
droplet[m].material = material;

group.add( droplet[n] );
```

4.4 液滴クラスの利用によるコードの効率化

2015 年に策定された ECMAScript［3］から JavaScript ではクラスが実装できるようになって，本格的なオブジェクト指向言語への仕様が固められた。本研究では，発生や分裂，消滅を繰り返す液滴を表すデータ構造としてクラスを採用した。

液滴クラスの設計（抜粋）

```
class Droplet extends THREE.Sprite {
    constructor( material ) {
        super( material );
        this.lifetime = 500;
        this.velocity = new THREE.Vector3();
        this.history2 = new Array();
    }

    //lifetime
    set lifetime( lifetime ) {
        this._lifetime = lifetime;
    }
    get lifetime() {
        return this._lifetime;
    }

    //velocity
    set velocity( velocity ) {
        this._velocity = velocity;
    }
    get velocity() {
        return this._velocity;
    }

    //history -> history2
    set history2( history2 ) {
        this._history2 = history2;
    }
    get history2() {
        return this._history2 ;
    }

    move() {
```

```
        if( this.lifetime > 0 ) {

            //successful!
            var p = new THREE.Vector3();
            p.copy( this.position );
            this.history2.push( p );

            this.position.x += this.velocity.x;
            this.position.y += this.velocity.y;
            this.position.z += this.velocity.z;

            this.velocity.y += -9.8 / 100;

            this.lifetime--;

            if( this.position.y < -500 ){
                this.lifetime = 0;
            }
        }
    }
}
```

4.5 結果

以上の設計を経て実装されたプログラムを実行すると，線香花火のシミュレーションを見ることができる。実行例の一部は，**図 5** の通りである。

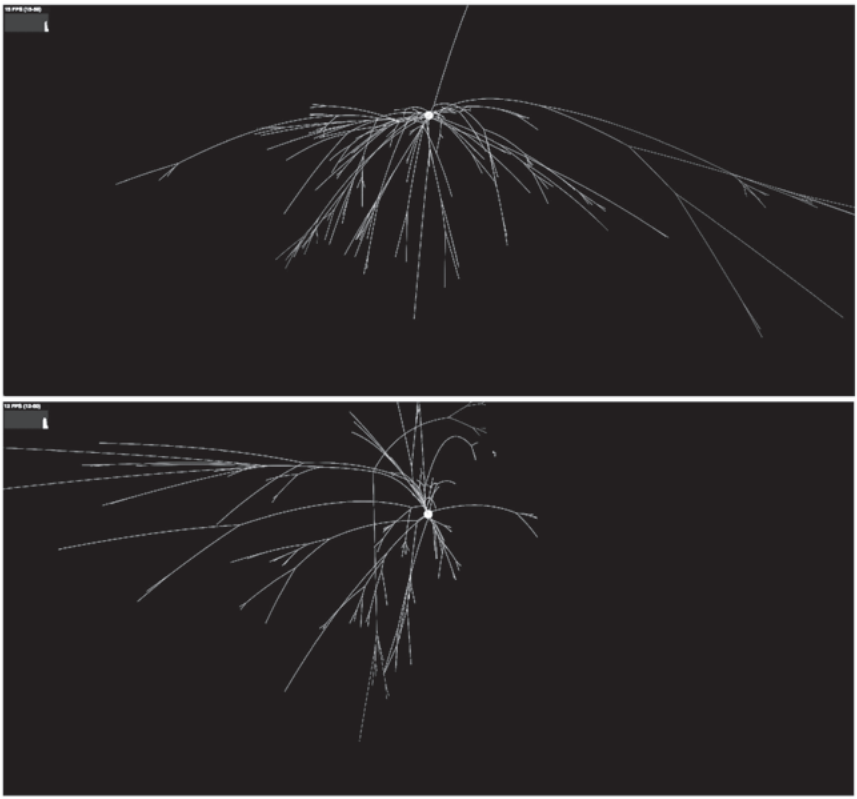

図 5　線香花火の CG 表現（アニメーション）

5　ま　と　め

本稿は，2015 年度に交付を受けた松山大学特別研究助成「偏微分方程式を用いた写実的な CG 映像表現に関する研究」によって写実的な CG 映像表現について研究を進めたものである。具体的な例題として，ホタルの乱舞，打ち上げ花火および線香花火の CG シミュレーションを開発した。

現在，CG に関する開発環境は低価格化によって導入しやすくなっており，例えば HTML5 Canvas 2D コンテキストや WebGL の実装でスマートフォンを

含む多くのデバイスで実行可能になっている。大きなコストをかけることなく，プログラム開発などの研究を遂行できる状態に至ってはいるものの，偏微分方程式など物理法則から第一原理計算を通じて映像表現の実装にまで至るような成果は多くない。新しい映像表現手法を理論的に提案するだけでなく，実装して表現するところまで研究を遂行することを目指した。

このような研究課題では，（主にコンピューティング環境における計算コストの理由から）それまで映像表現の計算方法として採用されることの少なかった偏微分方程式を用いることに学術的な特色・独創的な点があると考えられる。偏微分方程式を解析することで新しい映像表現を提案できるだけでなく，映像効果を実現するために適切な偏微分方程式を選択することにも当該研究分野に大きく貢献できる可能性がある。

最後に，本研究課題に取り組んで得られた最新の成果は，SIGGRAPH を含むトップレベルの国際会議に論文が採択される水準まで研究のクオリティを高めたい。そして，2016 年 8 月のゼミ合宿を企画し，愛媛県南予地区の自然造形という最高の楽しみを筆者に与えてくれた檀ゼミ 7 期生の学生たちに感謝するとともに，そのときの線香花火から得られたインスピレーションによって本稿が完成に至ることになったことを付記して締め括る。

参 考 文 献

［1］S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer（New York, 2002）; S. Osher and N. Paragios, *Geometric Level Set Methods in Imaging, Vision and Graphics*, Springer（New York, 2003）.

［2］HTML5 Canvas 2D Context, https://www.w3.org/TR/2dcontext/

［3］ECMA, http://www.ecma-international.org/ecma-262/6.0/

［4］WebGL - OpenGL ES for the Web, https://www.khronos.org/webgl/

［5］Osamu Shimomura, *Bioluminescence : Chemical Principles and Methods*, World Scientific Publishing（Singapore, 2006）.

［6］Albert Chern, Felix Knöppel, Ulrich Pinkall, Peter Schröder, and Steffen Weißmann, “Schrödinger’s smoke,” ACM Trans. Graph. 35, 4, Article 77（July 2016）, 13 pages. DOI :

https://doi.org/10.1145/2897824.2925868
https://www.youtube.com/watch?v=5C9BLAXCe1I
［7］Chihiro Inoue, Yu-ichiro Izato, Atsumi Miyake, and Emmanuel Villermaux, "Direct Self-Sustained Fragmentation Cascade of Reactive Droplets," Phys. Rev. Lett. 118, 074502－Published 14 February 2017.
［8］オリジナルの文献のほか，最近では，寺田寅彦「科学者とあたま」，平凡社（2015）および中谷宇吉郎「雪を作る話」，平凡社（2016）が入手可能である。
（以上，URL は 2017 年 3 月 25 日閲覧）

付　　録

ホタルの乱舞（ソースコード）

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>ホタルの乱舞</title>
<style>canvas{
    background-color: black;
}
</style>
<script>
var x  = new Array(); //ホタルの x 座標
var y  = new Array(); //ホタルの y 座標
var vx = new Array(); //ホタルの x 方向の速度(px/33ms = 30px/s)
var vy = new Array(); //ホタルの y 方向の速度(px/33ms = 30px/s)
var theta = new Array(); //ホタルの発光状態の位相
for( var j = 0; j < 16; j++ ){
    x[j]  = Math.random() * 640;
    y[j]  = Math.random() * 480;
    vx[j] = Math.random() * 3 - 1.5;
    vy[j] = Math.random() * 0.5 - 0.25;
    theta[j] = Math.random() * 2 * Math.PI;
}
function drawScreen(){
    var canvas = document.getElementById( "myCanvas" );
    var myContext = canvas.getContext( "2d" );

    // ホタルの位置を計算する
    for( var i = 0; i < 16; i++ ){
        vx[i] += ( Math.random() * 2 - 1 ) * 0.2;
        if( vx[i] < -3 || 3 < vx[i] ){
            vx[i] / = 2;
        }

        vy[i] += ( Math.random() * 2 - 1 ) * 0.1;
        if( vy[i] < -0.5 || 0.5 < vy[i] ){
```

```
            vy[i] /= 2;
        }

        x[i] = ( x[i] + vx[i] + 640 ) % 640;
        y[i] = ( y[i] + vy[i] + 480 ) % 480;
    }

    // ホタルの発光を描画する
    myContext.clearRect( 0, 0, 640, 480 );
    for( var i = 0; i < 16; i++ ){
        var a = Math.floor( ( Math.sin( theta[i] ) + 1 ) * 256 ) / 256;
        theta[i] += 0.05;

        var g
        = myContext.createRadialGradient( x[i], y[i], 0, x[i], y[i], 16 );
        g.addColorStop(  0/10, "rgba( 128, 255, 0, " + a + " )" );
        g.addColorStop(  1/10, "rgba( 128, 255, 0, " + a + " )" );
        g.addColorStop( 10/10, "rgba(   0,   0, 0, " + 0 + " )" );
        myContext.fillStyle = g;
        myContext.beginPath();
        myContext.arc( x[i], y[i], 16, 0, 2 * Math.PI, true);
        myContext.fill();
    }

    setTimeout( 'drawScreen()', 33 );
}
</script>
</head>
<body onload="drawScreen();">
<canvas id="myCanvas" width="640" height="480">
このブラウザは canvas 要素に対応していません。
</canvas>
</body>
</html>
```

http://www.cc.matsuyama-u.ac.jp/~dan/CG/work05.html

打ち上げ花火（ソースコード）

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>打ち上げ花火</title>
<style>
canvas{
    background-color: #000000;
}
</style>
<script>
var x     = new Array(); //花火パーティクルの x 座標
var y     = new Array(); //花火パーティクルの y 座標
var vx    = new Array(); //花火パーティクルの x 方向の速度
var vy    = new Array(); //花火パーティクルの y 方向の速度
var color = new Array(); //花火パーティクルの色
var cx    = Math.random() * 512 + 256; //花火中心点の x 座標
var cy    = Math.random() * 256; //花火中心点の y 座標
for( var j = 0; j < 512; j++ ){
    x[j] = cx;
    y[j] = cy;
    var speed = 4 * Math.random();
    var theta = Math.random() * 2 * Math.PI;
    vx[j] = speed * Math.cos( theta );
    vy[j] = speed * Math.sin( theta );
    var red = Math.floor( Math.random() * 128 ) + 128;
    color[j]="rgb("+red+",128,128)" ;
}

function drawScreen(){
    var canvas = document.getElementById( "myCanvas" );
    var myContext = canvas.getContext( "2d" );

    var i;

    for( i = 0; i < 512; i++ ){
```

```
        //vx[i] += 0.05 * ( Math.random() * 2 - 1 );
        vy[i] += 0.05;
        x[i] += vx[i];
        y[i] += vy[i];
    }

    myContext.fillStyle = 'rgba( 0, 0, 0, 0.05 )';
    myContext.fillRect( 0, 0, 1024, 768 );

    myContext.strokeWidth = 4;
    for( i = 0; i < 512; i++ ){
        myContext.beginPath();
        myContext.moveTo( x[i], y[i] ) ;
        myContext.lineTo( x[i] + vx[i], y[i] + vy[i] );
        myContext.strokeStyle = color[i];
        myContext.stroke();
    }

    setTimeout( 'drawScreen()', 33 );
}
</script>
</head>
<body onload="drawScreen();">
<canvas id="myCanvas" width="1024" height="768">
このブラウザは canvas 要素に対応していません。
</canvas>
</body>
</html>
```

http://www.cc.matsuyama-u.ac.jp/~dan/CG/work04.html

線香花火（ソースコード）

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>sparkling fireworks</title>
<meta charset="utf-8">
```

```
<meta name="viewport" content="width=device-width, user-scalable=no,
minimum-scale=1.0, maximum-scale=1.0">
<script src="../build/three.js"></script>
<script src="js/renderers/Projector.js"></script>
<script src="js/renderers/CanvasRenderer.js"></script>
<script src="js/libs/stats.min.js"></script>
<!--
<script src="https://raw.githubusercontent.com/mrdoob/three.js/dev/
build/three.min.js"></script>
<script src="https://raw.githubusercontent.com/mrdoob/three.js/dev/
examples/js/renderers/Projector.js"></script>
<script src="https://raw.githubusercontent.com/mrdoob/three.js/dev/
examples/js/renderers/CanvasRenderer.js"></script>
<script src="https://raw.githubusercontent.com/mrdoob/three.js/dev/
examples/js/libs/stats.min.js"></script>
-->
<style>
body{
    background-color: #000000;
    margin: 0px;
    overflow: hidden;
}
</style>
</head>
<body>
<script>
    var container, stats;
    var camera, scene, renderer, group;
    var done = false;
    var droplet = new Array();
    var mouseX = 0, mouseY = 0;

    var windowHalfX = window.innerWidth / 2;
    var windowHalfY = window.innerHeight / 2;
    var center = new THREE.Vector3(0, 500, 0 );

    const PI = Math.PI;
```

```
const PI2 = PI * 2;
var program = function( context ) {
    context.beginPath();
    context.arc( 0, 0, 0.5, 0, PI2, true);
    context.fill();
};

class Droplet extends THREE.Sprite {
    constructor( material ) {
        super( material );
        this.lifetime = 500;
        this.velocity = new THREE.Vector3();
        this.history2 = new Array();
    }

    //lifetime
    set lifetime( lifetime ) {
        this._lifetime = lifetime;
    }
    get lifetime() {
        return this._lifetime;
    }

    //velocity
    set velocity( velocity ) {
        this._velocity = velocity;
    }
    get velocity() {
        return this._velocity;
    }

    //history -> history2
    set history2( history2 ) {
        this._history2 = history2;
    }
    get history2() {
        return this._history2;
```

```
        }

        move() {
            if( this.lifetime > 0 ){
                var p = new THREE.Vector3();
                p.copy( this.position );
                this.history2.push( p );

                this.position.x += this.velocity.x;
                this.position.y += this.velocity.y;
                this.position.z += this.velocity.z;

                this.velocity.y += -9.8 / 100;

                this.lifetime--;

                if( this.position.y < -500){
                    this.lifetime = 0;
                }
            }
        }
    }

    init();
    animate();

    function init() {
        container = document.createElement( 'div' );
        document.body.appendChild( container );

        camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 1, 3000 );
        camera.position.z = 1500;

        scene = new THREE.Scene();

        group = new THREE.Group();
```

```
        scene.add(group);

        var material = new THREE.SpriteCanvasMaterial( {
            color: 0xff6633,
            program: program
        } );

        var particle = new THREE.Sprite( material );
        particle.position.copy( center );
        particle.scale.x = 50;
        particle.scale.y = 50;
        group.add( particle );

        renderer = new THREE.CanvasRenderer();
        renderer.setPixelRatio( window.devicePixelRatio );
        renderer.setSize( window.innerWidth, window.innerHeight );
        container.appendChild( renderer.domElement );

        stats = new Stats();
        container.appendChild( stats.dom );

        document.addEventListener( 'mousemove', onDocumentMouseMove,
false );
        document.addEventListener( 'touchstart', onDocumentTouchStart,
false );
        document.addEventListener( 'touchmove', onDocumentTouchMove,
false );
        document. addEventListener ( 'keydown', onDocumentKeyDown,
false );
        window.addEventListener( 'resize', onWindowResize, false ) ;
    }

    function onWindowResize() {
        windowHalfX = window.innerWidth / 2;
        windowHalfY = window.innerHeight / 2;

        camera.aspect = window.innerWidth / window.innerHeight;
```

```
    camera.updateProjectionMatrix();

    renderer.setSize( window.innerWidth, window.innerHeight );
}

function onDocumentMouseMove( event ) {
    mouseX = event.clientX - windowHalfX;
    mouseY = event.clientY - windowHalfY;
}

function onDocumentKeyDown( event ) {
    if( event.keyCode == 0x20 ) {

        var material = new THREE.LineBasicMaterial({
            color: 0xff0000,
            linewidth: 1
        });

        for( var i = 0; i < droplet.length; i++ ){

            var geometry = new THREE.Geometry();
            for(var j = 0; j < droplet[i].history2.length; j++ ){
                geometry.vertices.push( droplet[i].history2[j] );
            }

            var line = new THREE.Line( geometry, material );
            group.add( line ) ;

        }
    }
}

function onDocumentTouchStart( event ) {

    if( event.touches.length === 1 ) {

        event.preventDefault();
```

```
        mouseX = event.touches[ 0 ].pageX - windowHalfX;
        mouseY = event.touches[ 0 ].pageY - windowHalfY;

    }
}

function onDocumentTouchMove( event ) {
    if( event.touches.length === 1 ) {

        event.preventDefault();

        mouseX = event.touches[ 0 ].pageX - windowHalfX;
        mouseY = event.touches[ 0 ].pageY - windowHalfY;
    }
}

function animate() {
    requestAnimationFrame( animate );

    render();
    stats.update();
}

function render() {
    camera.position.x += (   mouseX - camera.position.x ) * 0.05;
    camera.position.y += ( - mouseY - camera.position.y ) * 0.05;
    camera.lookAt( scene.position );

    // droplet emission from the main drop
    if( !done && Math.random() * 100 < 1 ){

        //done = true;
        var material = new THREE.SpriteCanvasMaterial( {
            color: 0xffcc99,
            program: program
        } );
        for(var i = 0; i < 10 * Math.random(); i++ ) {
```

```
        droplet.push( new Droplet( material ) );
        var n = droplet.length - 1;
        droplet[n].position.copy( center );

        var s = PI  * Math.random();
        var t = PI2 * Math.random();
        droplet[n].velocity.x = 10 * Math.sin( s ) * Math.sin( t );
        droplet[n].velocity.y = 10 * Math.sin( s ) * Math.cos( t );
        droplet[n].velocity.z = 10 * Math.cos( s );

        var x, y, z;
        do{
            x = 2.0 * Math.random() - 1.0;
            y = 2.0 * Math.random() - 1.0;
            z = 2.0 * Math.random() - 1.0;
        }while( x * x + y * y + z * z > 1.0 );

        droplet[n].velocity.x = 10 * x;
        droplet[n].velocity.y = 10 * y;
        droplet[n].velocity.z = 10 * z;

        droplet[n].scale.x = 8;
        droplet[n].scale.y = 8;
        group.add( droplet[n] );

    }
}

for( var i = 0; i < droplet.length; i++ ){

    if( droplet[i].lifetime > 0 ){
        droplet[i].move();
    }
    else{
        droplet[i].visible = false;
        group.remove( droplet[i] );
    }
```

```
}

if( droplet.length > 0 ){

    var material = new THREE.SpriteCanvasMaterial( {
        color: 0xffffff,
        program: program
    } );

    var n = droplet.length;
    var m;

    do{
        m = Math.floor( n * Math.random() );
    }while( droplet[m].visible == false );

    droplet.push( new Droplet(material) );
    droplet[n].position.copy( droplet[m].position );

    droplet[m].scale.x /= Math.cbrt(2.0);
    droplet[m].scale.y /= Math.cbrt(2.0);
    droplet[n].scale.x = droplet[m].scale.x;
    droplet[n].scale.y = droplet[m].scale.y;

    droplet[m].lifetime /= 2;
    droplet[n].lifetime = droplet[m].lifetime;

    var s = PI  * Math.random();
    var t = PI2 * Math.random();
    var vx = droplet[n].scale.x * Math.sin( s ) * Math.sin( t );
    var vy = droplet[n].scale.x * Math.sin( s ) * Math.cos( t );
    var vz = droplet[n].scale.x * Math.cos( s );

    droplet[n].velocity.x = droplet[m].velocity.x + vx;
    droplet[n].velocity.y = droplet[m].velocity.y + vy;
    droplet[n].velocity.z = droplet[m].velocity.z + vz;
```

```
            droplet[m].velocity.x = droplet[m].velocity.x - vx;
            droplet[m].velocity.y = droplet[m].velocity.y - vy;
            droplet[m].velocity.z = droplet[m].velocity.z - vz;
            droplet[m].material = material;

            group.add( droplet[n] );

        }

        renderer.render( scene, camera );

    }
</script>
</body>
</html>
```