

# FireFlower

P. Baillehache

October 1, 2017

## Contents

<b>1</b>	<b>Makefile</b>	<b>1</b>
<b>2</b>	<b>Usage</b>	<b>2</b>

## Introduction

FireFlower is C program creating firework-like pattern.

The pattern is made of a set of particles, each trying to follow two others, and having a different color. The particle move at constant speed toward there targets with an attraction force proportional to the distance to the target. The particles leave a trace of their color, which progressively disappear. The size of the particle, length of the tail, inertia, and number of particles is either set by the user, or randomly set.

## 1 Makefile

```
OPTIONS_DEBUG=-ggdb -g3 -Wall
OPTIONS_RELEASE=-O3
OPTIONS=$(OPTIONS_RELEASE)
INCPATH=/home/bayashi/Coding/Include
LIBPATH=/home/bayashi/Coding/Include

all : main

main: main.o Makefile $(LIBPATH)/tgapaint.o $(LIBPATH)/gset.o $(LIBPATH)/pbmath.o $(LIBPATH)/bcurve.o
gcc $(OPTIONS) main.o $(LIBPATH)/tgapaint.o $(LIBPATH)/pbmath.o $(LIBPATH)/gset.o $(LIBPATH)/bcurve.o -o main -lm

main.o : main.c Makefile
gcc $(OPTIONS) -I$(INCPATH) -c main.c
```

```

clean :
rm -rf *.o main

test :
main out.tga -rnd

valgrind :
valgrind -v --track-origins=yes --leak-check=full --gen-suppressions=yes --show-leak-kinds=all ./main

video:
avconv -r 20 -i ./Frames/frame%03d.tga -b:v 2048k video.mp4

```

## 2 Usage

```

// ===== SMOKER.C =====

// ===== Include =====

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include "gset.h"
#include "pbmath.h"
#include "tgapaint.h"

// ===== Define =====

#define SCREENSIZE 1000
#define rnd() (float)(rand()/(float)(RAND_MAX))

// ===== Structure =====

struct Particle;
typedef struct Particle {
    // Position
    VecFloat *_pos;
    // Speed
    VecFloat *_speed;
    // Pencil
    TGAPencil *_pen;
    // Targets
    struct Particle **_targets;
} Particle;

// ===== Functions =====

Particle* ParticleCreate(int nbTarget, int thickness) {
    // Allocate memory
    Particle *p = (Particle*)malloc(sizeof(Particle));
    // If we could allocate memory
    if (p != NULL) {
        // Set properties
        p->_pos = VecFloatCreate(2);
        if (p->_pos == NULL)
            return NULL;
        p->_speed = VecFloatCreate(2);
        if (p->_speed == NULL)

```

```

        return NULL;
    p->_pen = TGAGetBlackPencil();
    if (p->_pen == NULL)
        return NULL;
    TGAPencilSetAntialias(p->_pen, true);
    TGAPencilSetThickness(p->_pen, thickness);
    TGAPencilSetShapeRound(p->_pen);
    p->_targets = (Particle**)malloc(sizeof(Particle*) * nbTarget);
    if (p->_targets == NULL)
        return NULL;
}
// Return the particle
return p;
}

void ParticleFree(Particle **p) {
    VecFree(&((*p)->_pos));
    VecFree(&((*p)->_speed));
    TGAFreePencil(&((*p)->_pen));
    free(*p);
    *p = NULL;
}

void FireFlower(char *fileName, int nbParticle, float fading,
    float speed, float inertia, int nbTarget, float thickness) {
    // Create the TGA
    VecShort *dim = VecShortCreate(2);
    if (dim == NULL) {
        fprintf(stderr, "VecShortCreate failed\n");
        return;
    }
    VecSet(dim, 0, SCREENSIZE);
    VecSet(dim, 1, SCREENSIZE);
    TGA_Pixel *pixelBg = TGAGetBlackPixel();
    if (pixelBg == NULL) {
        fprintf(stderr, "TGAGetBlackPixel failed\n");
        return;
    }
    TGA *tga = TGACreate(dim, pixelBg);
    if (tga == NULL) {
        fprintf(stderr, "TGACreate failed\n");
        return;
    }
    // Create the particles
    GSet *particles = GSetCreate();
    if (particles == NULL) {
        fprintf(stderr, "GSetCreate failed\n");
        return;
    }
    for (int iParticle = nbParticle; iParticle--;) {
        Particle *p = ParticleCreate(nbTarget, thickness);
        if (p == NULL) {
            fprintf(stderr, "ParticleCreate failed\n");
            return;
        }
        for (int dim = 2; dim--;)
            VecSet(p->_pos, dim, (0.1 + rnd() * 0.8) * (float)SCREENSIZE);
        TGA_Pixel *color = TGAGetWhitePixel();
        color->_rgba[4] = 125;
        for (int irgb = 3; irgb--;)
            color->_rgba[irgb] = (unsigned char)floor(rnd() * 255.0);
        TGAPencilSetColor(p->_pen, color);
    }
}

```

```

    TGAFreePixel(&color);
    GSetAppend(particles, p);
}
// Set the target of the particles
GSetElem *elem = particles->_head;
int iElem = 0;
while (elem != NULL) {
    Particle *p = (Particle*)(elem->_data);
    for (int iTarget = nbTarget; iTarget--;) {
        int jElem = iElem;
        while (iElem == jElem)
            jElem = (int)floor(rnd() * (float)nbParticle);
        p->_targets[iTarget] =
            (Particle*)GSetGet(particles, jElem);
    }
    ++iElem;
    elem = elem->_next;
}
// Declare a flag to stop the simulation loop
bool flagStop = false;
// Declare a vector equal to the centre of the image
VecFloat *center = VecFloatCreate(2);
if (center == NULL) {
    fprintf(stderr, "VecFloatCreate failed\n");
    return;
}
VecSet(center, 0, 0.5 * (float)SCREENSIZE);
VecSet(center, 1, 0.5 * (float)SCREENSIZE);
// Simulation loop
float t = 0.0;
int iFrame = 0;
int nbImpact = 0;
while (flagStop == false) {
    fprintf(stderr, "%.1f      \r", t);
    // Fading
    VecShort *pos = VecShortCreate(2);
    if (pos == NULL) {
        fprintf(stderr, "VecShortCreate failed\n");
        return;
    }
    for (VecSet(pos, 0, 0); VecGet(pos, 0) < SCREENSIZE;
         VecSet(pos, 0, VecGet(pos, 0) + 1)) {
        for (VecSet(pos, 1, 0); VecGet(pos, 1) < SCREENSIZE;
             VecSet(pos, 1, VecGet(pos, 1) + 1)) {
            TGAPixel *pixel = TGAGetPix(tga, pos);
            TGAPixel *blend = TGABlendPixel(pixel, pixelBg, fading);
            TGASetPix(tga, pos, blend);
            TGAFreePixel(&blend);
        }
    }
    VecShortFree(&pos);
    // For each particle
    elem = particles->_head;
    iElem = 0;
    while (elem != NULL) {
        Particle *p = (Particle*)(elem->_data);
        // For each target
        for (int iTarget = nbTarget; iTarget--;) {
            Particle *pT = p->_targets[iTarget];
            // Get the attraction
            VecFloat *v = VecGetOp(pT->_pos, 1.0, p->_pos, -1.0);
            float d = VecNorm(v);

```

```

float a = 1.0 / d;
if (d < 0.5) {
    // Impact, change the target
    ++nbImpact;
    if (nbImpact == 5)
        flagStop = true;
    int jElem = iElem;
    while (iElem == jElem)
        jElem = (int)floor(rnd() * (float)nbParticle);
    p->_targets[iTarget] =
        (Particle*)GSetGet(particles, jElem);
}
VecNormalise(v);
// Update speed
VecOp(p->_speed, inertia, v, a);
VecNormalise(p->_speed);
VecOp(p->_speed, speed, NULL, 1.0);
// Free memory
VecFree(&v);
}
// Add attraction toward the center of the image
VecFloat *v = VecGetOp(center, 1.0, p->_pos, -1.0);
float d = VecNorm(v);
//float a = 1.0 / (d * (float)nbTarget);
float a = 1.0 * pow(d / (float)SCREENSIZE, 3.0);
VecNormalise(v);
// Update speed
VecOp(p->_speed, inertia, v, a);
VecNormalise(p->_speed);
VecOp(p->_speed, speed, NULL, 1.0);
// Free memory
VecFree(&v);
// Draw
VecFloat *to = VecGetOp(p->_pos, 1.0, p->_speed, 1.0);
if (to == NULL) {
    fprintf(stderr, "VecGetOp failed\n");
    return;
}
TGADrawLine(tga, p->_pos, to, p->_pen);
elem = elem->_next;
++iElem;
}
// Move particles
elem = particles->_head;
while (elem != NULL) {
    Particle *p = (Particle*)(elem->_data);
    VecOp(p->_pos, 1.0, p->_speed, 1.0);
    elem = elem->_next;
}
// Save the frame
//char frame[100];
//sprintf(frame, "./Frames/frame%03d.tga", iFrame);
//TGASave(tga, frame);
// End condition
t += 1.0;
++iFrame;
if (t * 2.0 > SCREENSIZE)
    flagStop = true;
}
fprintf(stderr, "\n");
// Save the TGA

```

```

int ret = TGASave(tga, fileName);
if (ret != 0) {
    fprintf(stderr, "TGASave failed\n");
    return;
}
// Free memory
elem = particles->_head;
while (elem != NULL) {
    ParticleFree((Particle**>(&(elem->_data)));
    elem = elem->_next;
}
GSetFree(&particles);
VecFree(&dim);
TGAFreePixel(&pixelBg);
TGAFree(&tga);
}

// ===== Main function =====

int main(int argc, char **argv) {
    // Initialize the random generator
    time_t seed = time(NULL);
    srand(seed);
    // Declare variables to memorize the arguments and set default values
    if (argc < 2)
        return 1;
    char *fileName = argv[1];
    int nbParticle = 100;
    float fading = 0.05;
    float speed = 5.0;
    float inertia = 0.01;
    int nbTarget = 3;
    float thickness = 2.0;
    // Decode arguments
    for (int iArg = 2; iArg < argc; ++iArg) {
        if (strcmp(argv[iArg], "-rnd") == 0) {
            thickness = 1.0 + rnd() * 4.0;
            inertia = rnd() * 0.03;
            fading = 0.01 + 0.09 * rnd();
            nbParticle = 25 + (int)floor(rnd() * 75.0);
            fprintf(stdout, "thickness: %.3f\n", thickness);
            fprintf(stdout, "inertia: %.3f\n", inertia);
            fprintf(stdout, "fading: %.3f\n", fading);
            fprintf(stdout, "nbParticle: %d\n", nbParticle);
        } else if (strcmp(argv[iArg], "-thick") == 0 && iArg + 1 < argc) {
            ++iArg;
            sscanf(argv[iArg], "%f", &thickness);
        } else if (strcmp(argv[iArg], "-inertia") == 0 && iArg + 1 < argc) {
            ++iArg;
            sscanf(argv[iArg], "%f", &inertia);
        } else if (strcmp(argv[iArg], "-fading") == 0 && iArg + 1 < argc) {
            ++iArg;
            sscanf(argv[iArg], "%f", &fading);
        } else if (strcmp(argv[iArg], "-nb") == 0 && iArg + 1 < argc) {
            ++iArg;
            sscanf(argv[iArg], "%d", &nbParticle);
        } else if (strcmp(argv[iArg], "-help") == 0) {
            printf("arguments : <filename> [-help] [-rnd] ");
            printf("[-thick] [-inertia] [-fading] [-nb]\n");
            // Stop here
            return 0;
        }
    }
}

```

```

}
// Create the image
FireFlower(fileName, nbParticle, fading, speed, inertia, nbTarget,
    thickness);
// Return the success code
return 0;
}

```

