ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КВАЛІФІКАЦІЙНА РОБОТА

НА ТЕМУ: Методи шифрування інформації за допомогою алгебраїчних фракталів

ВИКОНАВ:
Студент гр. СПзм-20-1 Караджян Б. Ю.

КЕРІВНИК:
к.т.н. доц. Ільїна І.В.

ХАРКІВ
2022р.

Мета та завдання

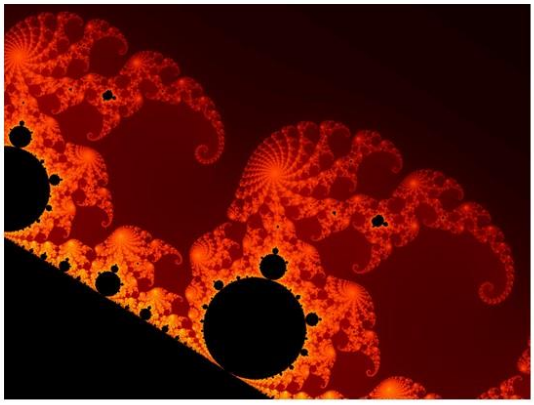
Метою кваліфікаційної роботи є розробка методів шифрування інформації на основі алгебраїчних фракталів

У ході виконання кваліфікаційної роботи були вирішені наступні завдання:

- проаналізовано сфери застосування фрактарів;
- проаналізовано типи алгебраїчних фракталів;
- розроблено алгоритм шифрування зображень;
- розроблено алгоритм шифрування тексту.

Термінологія

Фракта́л (від лат. *fractus* — подрібнений, дробовий) — у побутовому розумінні часто означають як деяку нерегулярну, самоподібну структуру. Більш строге означення фрактала вимагає глибоких знань із курсів алгебри і математичного аналізу. Поширеним є розуміння фрактала як множини, яка має властивість самоподібності, тобто такої множини, що складається з частин, які є подібними до неї самої.



3

Деякі сфери застосування фракталів

- Фрактальні криві та поверхні
- Фрактальна графіка та обробка зображень
- Фрактальна криптографі
- Фрактали у технічних науках

4

Метод шифрування зображень за допомогою фракцій алгебри

1. Зображення має бути представлене потоком бітів. (Кожний піксель займає 3 байти пам'яті і його можна представити у вигляді беззнакового цілого в діапазоні від 0 до 16777215, а потім це значення переводиться в 24-бітне двійкове число.)

2. Ключ формується так:

2.1 вибирається вид фракції алгебри;

2.3 якщо вибрано множину Жюлі, то задається константа;

2.3 вказується кількість ітерацій та параметри для функції кольору;

2.4 визначається масштаб (що більша кількість ітерацій і чим сильніше наближення, тим цікавіша та складніша структура фракталу, що добре відображається на шифруванні) та розміри шифруючого зображення (вони повинні співпадати з вихідним зображенням).

3. На основі цих даних створюється шифруюче зображення, яке так само, як і шифрується, представляється у вигляді потоку бітів.

4. Виконується операція XOR між зображенням, що шифрується і шифрує, і потік бітів назад переводиться в пікселі, виходить первинне зашифроване зображення.

5. До отриманого зображення застосовується алгоритм дифузії, який можна подати за такою формулою:

$$c_i = c_{i-1} \oplus c'_{i+1}, i = 1,..., N, \qquad (9)$$

де c - це пікселі підсумкового зашифрованого повідомлення, а c' - первинного, N– загальна кількість пікселів.

5

Метод шифрування текстів за допомогою алгебраїчних фракталів

1. Кожен символ тексту займає 1 байт пам'яті, тобто його можна уявити двійковою восьмибітовою послідовністю.

2. Ключ формується так само, як і при шифруванні зображення (тут, кількість пікселів має бути більшою або дорівнює кількості символів у тексті).

3. Оскільки піксель кодується 3 байтами, а символ лише одним (і тому зробити операцію XOR стає проблематично) можливі наступні варіанти:

3.1 піксель шифруючого зображення відповідно до RGB моделі розпадається на три компоненти по одномубайту: Red(r_i), Green(g_i) і Blue(b_i), i = 0,...,7 . Далі виконується логічна операція AND (I) між непарними бітами Red і Green, і між парними бітами Red і Blue. Цей процес відображено у формулі (10):

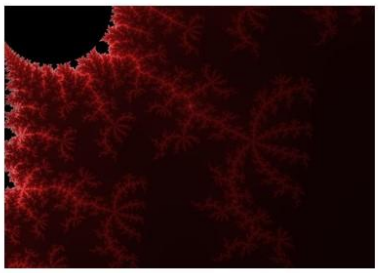
$$a_i = \begin{cases} r_i \,\&\, g_i, i = 1,3,5,7 \\ r_i \,\&\, b_i, i = 0,2,4,6 \end{cases}$$

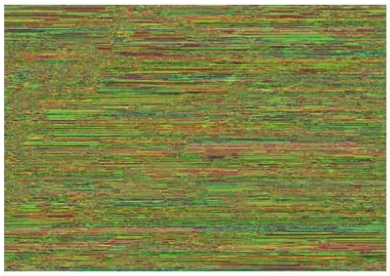
3.2 виконується операція XOR міжбітами Red(ri), Green(gi) і Blue(bi), i = 0,...,7.

4. Завдяки проведеним перетворенням, виходить необхідна шифруюча послідовність, між якою і потоком бітів тексту, що шифрується, проводиться операція XOR. Потім біти переводять у символи, і виходить шифротекст.

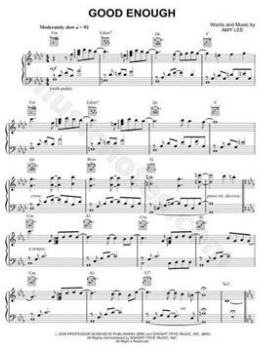
6

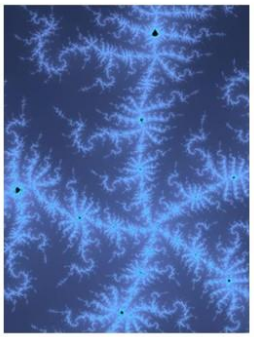
Шифрування та дешифрування зображень

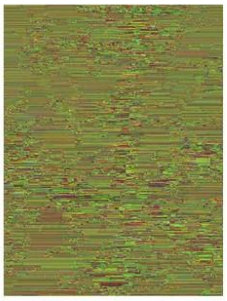


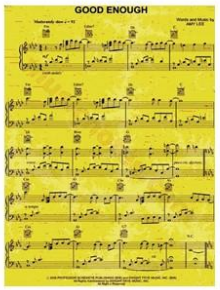


7





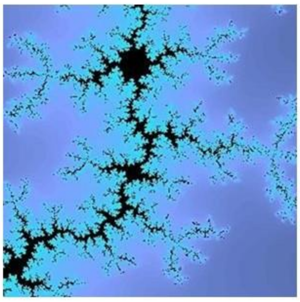




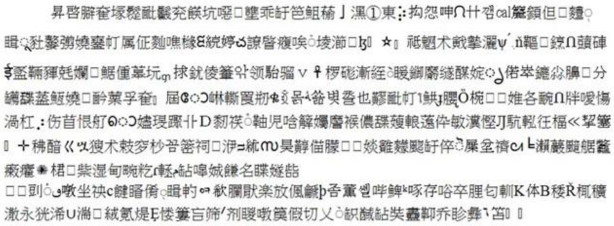
8

Шифрування та дешифрування текстів

"Curiouser and curiouser!" cried Alice (she was so much surprised, that for the moment she quite forgot how to speak good English); "now I'm opening out like the largest telescope that ever was! Good-bye, feet!" (for when she looked down at her feet, they seemed to be almost out of sight, they were getting so far off). "Oh, my poor little feet, I wonder who will put on your shoes and stockings for you now, dears? I'm sure I shan't be able! I shall be a great deal too far off to trouble myself about you: you must manage the best way you can; --but I must be kind to them," thought Alice, "or perhaps they won't walk the way I want to go! Let me see: I'll give them a new pair of boots every Christmas."



oF;u8n$s$rpb?`uf%v9n'r7ss#pg&o3bvQ8m7gt8|0u0y{HtAE*тщі*ен¬ PANH-OE\Z[OPUGEjF-V0s$ ио4`&/BR Eq@G/4d%ngм+;(; *;ou I&m opeNxQ'c'=7p,?IL 8A/▯ 5;0d&EEBQ@]PIAcpO©aye¬ зee5K0Hvb@qc8&%`5 <o>k4dpd+w> 1td`%v`b.d<-it!e0 1a-m&dcv, *dig,m+s] xtV E8pm@4!`wG%D'a4r9n7 #opf1rpn4d{,r -h} =ytt;k'$8mvuBvu8FY¤щuM KowQppTJLOV[S-OSYAEXRFJO_BFbpe3u .?_t`7Un+%@l}{c3%0) 1(.sH be abhe!©hIy\,` #t}x&#MbrDu!`0E A-DHPTKbeHI31tВнщф8|.IheA}c&geart8eqb4q$ 'h=_ )o7%#d.?j,dc<tilim>p?#)alh#j($8jmp%e-T,J17e#l)!HeBM E,e4wdr"m) 'g$/> 'c<kqt8eqw0yrM@sj$nBw 1F\`E}0QkrI<'@U E E ARXO-HSVR-4@f?5=a'.H



Висновок

В результаті виконаної роботи було розроблено криптографічні алгоритми шифрування зображень та текстів з використанням теорії фрактального моделювання. На основі запропонованих алгоритмів у середовищі С++ були написані програми, що реалізують побудову фракцій алгебри та шифрування графічної та текстової інформації за допомогою них.

Результати роботи програм показали, що використовувати фрактали як ключ для шифрування досить зручно і ефективно. Для їх побудови потрібна невелика кількість параметрів, при цьому на виході виходять об'єкти зі складними хаотичними межами.

Так як шифруюча послідовність будується на основі алгебраїчного фракталу, кращий результат можна досягти, якщо використовувати фрактали з досить порізаною структурою. Наприклад, фрагменти сильно наближеної на межі множини Мандельброта або нескладне безліч Жюлія.

При дешифрації зображень з'являється жовтий колір і залишкові сліди від фракталу, але зображені об'єкти, інформація, представлена у графічному вигляді, зберігаються. Це добре видно на прикладі нотної сторінки

ДОДАТОК Б

Лістинг програми для побудови фракцій алгебри

```
#version 120
uniform float screen_ratio;
uniform float scale;
uniform vec2 translation;
uniform bool julia_enabled;
uniform float time;
const int depth = 500;
float sqr(float val)
{
52
return val * val;
}
vec2 sqr_complex(vec2 complex)
{
//(a+bi)^2 = a^2 - b^2 + 2abi
vec2 result;
result.x = sqr(complex.x) - sqr(complex.y);
result.y = 2.0 * complex.x * complex.y;
//result.x = (2/3.f)*complex.x + (complex.x*complex.x-
complex.y*complex.y) /
(3.f*sqr(complex.x*complex.x + complex.y*complex.y));
//result.y = (2 / 3.f)*complex.y *(1-
complex.y/sqr(complex.x*complex.x + complex.y*complex.y));
return result;
}
vec2 next_z(vec2 prev_z, vec2 c){
return sqr_complex(prev_z) + c;
}
vec4 to_color(float value)
{
struct stop
{
float offset;
vec3 color;
};
stop stops[] = stop[](
stop(0, vec3(0.0, 0.0, 0.0)),
stop(0.5, vec3(0.5, 0.625, 1.0)),
stop(1.0, vec3(0.0, 1.0, 1.0))
);
for(int i = 1; i < stops.length(); ++i)
{
stop prev_stop= stops[i - 1];
stop cur_stop = stops[i];
if(value <= cur_stop.offset && value >= prev_stop.offset)
{
float balance = (value - prev_stop.offset) / (cur_stop.offset -
prev_stop.offset);
```

```
vec3 color = (1.0 - balance) * prev_stop.color + balance *
cur_stop.color;
return vec4(color, 1.0);
}
}
}
void main()
{
vec2 c = (gl_TexCoord[0].st * 2.0 - vec2(1.0));
53
c *= scale;
c += translation;
c.x *= screen_ratio;

vec2 z = c;
if(julia_enabled)
{
float radius = sin(time * 10.0) * 0.5 + 1.0;
float angle = time * 10.0;
c = vec2(cos(angle), sin(angle)) * radius;
}
int i;
 for(i = 0; i < depth; i++)
{
z = next_z(z, c);
 if(dot(z,z) > 4.0) break;
//dot(z,z) = |z|*|z|*cos(alpha) = z.x*z.x + z.y*z.y = |z|^2
 }
if(i == depth)
i = 0;
/*vec2 d=z, t;
i=0;
while ((dot(z,z) <
1000000)&&((sqr(d.x)+sqr(d.y))>0.0000001)&&(i<depth))
{
t=z;
z = sqr_complex(t);
d.x=t.x-z.x; if(d.x<0) d.x=-d.x;
d.y=t.y-z.y; if(d.y<0) d.y=-d.y;
i++;
}*/
 gl_FragColor = to_color(float(i) / float(depth));
}
vertex.vs (вершинный шейдер)
#version 120
void main()
{
gl_TexCoord[0] = gl_MultiTexCoord0;
gl_Position = gl_Vertex;
}
main.cpp
#define _CRT_SECURE_NO_WARNINGS
#include <exception>
```

```
#include <iostream>
#include <cassert>
54
#include "glut.h"
#include "extensions.h"
#include "ShaderObjects.h"
CShaderProgram* program = nullptr;
float screen_ratio = 1.0f;
float scale = 1.0f;
float translation[2] = { 0.0f, 0.0f };
bool julia_enabled = true;
void reshape(int width, int height)
{
glViewport(0, 0, width, height);
screen_ratio = float(width) / float(height);
}
void display(void)
{
try
{
glClear(GL_COLOR_BUFFER_BIT);
program->use_program();
program->set_uniform_float("screen_ratio", screen_ratio);
program->set_uniform_float("scale", scale);
program->set_uniform_vec2("translation", translation[0],
translation[1]);
program->set_uniform_int("julia_enabled", julia_enabled ? 1 :
0);
if (julia_enabled)
{
static float time = 0.0f;
program->set_uniform_float("time", time += 0.001f);
}
glBegin(GL_QUADS);
glTexCoord2f(0.0, 0.0);
glVertex2d(-1.0, -1.0);
glTexCoord2f(1.0, 0.0);
glVertex2d(1.0, -1.0);
glTexCoord2f(1.0, 1.0);
glVertex2d(1.0, 1.0);
glTexCoord2f(0.0, 1.0);
glVertex2d(-1.0, 1.0);
glEnd();
glutSwapBuffers();
glutPostRedisplay();
}
catch (std::exception& ex)
55
{
std::cout << ex.what() << std::endl;
assert
(false);
}
```

```
}
void keyboard(unsigned char key, int, int
)
{
switch
(key
)
{
case 27:
delete program;
exit(0);
break
;
case ' '
:
julia_enabled = !julia_enabled;
}
}
void special(int key, int, int
)
{
switch
(key
)
{
case GLUT_KEY_UP
:
translation[1] += 0.01f * scale;
break
;
case GLUT_KEY_DOWN
:
translation[1]
-= 0.01f * scale;
break
;
case GLUT_KEY_LEFT
:
translation[0]
-= 0.01f * scale;
break
;
case GLUT_KEY_RIGHT
:
translation[0] += 0.01f * scale;
break
;
case GLUT_KEY_PAGE_UP
:
scale *= 0.5;
break
;
case GLUT_KEY_PAGE_DOWN
```

```
:
scale *= 2.0;
break
;
}
}
void init() {
try
56
{
load_extensions();
program = new CShaderProgram("shaders/vertex.vs",
"shaders/fragment.fs");
}
catch (std::exception& ex)
{
std::cout << ex.what() << std::endl;
assert(false);
}
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitWindowSize(800, 600);
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
glutCreateWindow("fractals");
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutSpecialFunc(special);
init();
glutMainLoop();
return 0;
}
```

ДОДАТОК В

ЛІСТИНГ ПРОГРАМИ ДЛЯ ШИФРУВАННЯ

```
#define _SCL_SECURE_NO_WARNINGS
#define _CRT_SECURE_NO_WARNINGS
#include <boost/gil/image.hpp>
#include <boost/gil/typedefs.hpp>
#include <boost/gil/extension/io/jpeg_io.hpp>
#include <boost/gil/extension/io/png_io.hpp>
#include <iostream>
#include <cstdint>
using namespace boost;
64
struct img_vector_t
{
std::vector<uint32_t> data;
size_t width;
size_t height;
};
union UPixel
{
struct
{
uint8_t red;
uint8_t green;
uint8_t blue;
};
uint32_t color;
};
void load_image_as_vector(const char* file_name, img_vector_t&
result)
{
gil::rgb8_image_t img;
gil::png_read_image(file_name, img);
result.width = img.width();
result.height = img.height();
result.data.clear();
result.data.reserve(result.width * result.height);
class Functor
{
img_vector_t* m_pDestination;
public:
Functor(img_vector_t& destination) :
m_pDestination(&destination) {}
void operator()(gil::rgb8_pixel_t p)
{
auto red = gil::at_c<0>(p);
auto green = gil::at_c<1>(p);
auto blue = gil::at_c<2>(p);
UPixel test;
test.color = 0;//initialize union
test.red= red;
```

```
test.green = green;
test.blue = blue;
m_pDestination->data.push_back(test.color);
}
};
65
gil::for_each_pixel(gil::const_view(img), Functor(result));
}
std::auto_ptr<gil::rgb8_image_t>
convert_vector_to_image(img_vector_t& vec)
{
std::auto_ptr<gil::rgb8_image_t> pResult(new
gil::rgb8_image_t(vec.width, vec.height));
for (auto x = 0; x < vec.width; ++x)
{
auto it = pResult->_view.col_begin(x);
for (auto y = 0; y < vec.height; ++y)
{
UPixel pixel;
pixel.color = vec.data[y * vec.width + x];
gil::at_c<0>(it[y]) = pixel.red;
gil::at_c<1>(it[y]) = pixel.green;
gil::at_c<2>(it[y]) = pixel.blue;
}
}
return pResult;
}
size_t gamma(size_t a, size_t b)//гаммирование
{
if (a == 0 && b == 0) return 0;
if (a == 0 && b == 1) return 1;
if (a == 1 && b == 0) return 1;
if (a == 1 && b == 1) return 0;
}
void main()
{
img_vector_t fractal;
load_image_as_vector("fw.png", fractal);
img_vector_t plain;
load_image_as_vector("whale.png", plain);
size_t n, m = 24, **T, **F, **Ch,**Ch1;
int i, j;
n=plain.width*plain.height;
T = new size_t*[n];
for (i = 0; i<n; i++)
T[i] = new size_t[m];
size_t a;
for (i = 0; i<n; i++)
{
a = plain.data[i]; j = m - 1;
while (j >= 0)
{
if ((a == 0) || (a == 1)) { T[i][j] = 0; j--; }
```

```
else
{
66
if (a % 2 == 0) T[i][j] = 0;
else T[i][j] = 1;
a = a / 2; j--
;
}
}
}
F = new size_t*[n];
for (i = 0; i<n; i++)
F[i] = new size_t[m];
for (i = 0; i < n; i++) {
a = fractal.data
[
i
]; j = m
- 1;
while (j >= 0) {
if ((a == 0) || (a == 1)) { F[i][j] = 0; j--; }
else {
if (a % 2 == 0) F[i][j] = 0;
else F[i][j] = 1;
a = a / 2; j--
;
}
}
}
Ch1 = new size_t*[n];
for (i = 0; i<n; i++)
Ch1[i] = new size_t[m];
for (i = 0; i<n; i++) {
for (j = 0; j<m; j++) {
Ch1[i][j] = gamma(T[i][j], F[i][j]);
}
}
Ch = new size_t*[n];
for (i = 0; i<n; i++)
Ch[i] = new size_t[m];
for (j = 0; j < m; j++) {
Ch[0][j] = F[0][j];
Ch[n
- 1][j] = F[n
- 1][j];
}
for (i = 1; i<n
-1; i++)
{
for (j = 0; j<m; j++) {
Ch[i][j] = gamma(Ch1[i+1 ][j], Ch[i
- 1][j]);
}
```

```
}
67
img_vector_t ch;
ch.width = plain.width;
ch.height = plain.height;
load_image_as_vector("fw.png", ch);
uint32_t u;
for (i = 0; i<n; i++)
{
u = 0; double o = 23;
for (j = 0; j<m; j++)
{
u = u + Ch[i][j] * pow(2.0, o);
o = o - 1;
}
ch.data[i] = u;
}
///////////////
auto pImage = convert_vector_to_image(ch);
gil::png_write_view("C:\\Users\\Rinwen\\Desktop\\дешифр\\diploma
\\imagereader\\chwhale.png", gil::const_view(*pImage.get()));
for (i = 0; i<n; i++)
delete T[i];
delete[]T;
for (i = 0; i<n; i++)
delete F[i];
delete[]F;
for (i = 0; i<n; i++)
delete Ch[i];
delete[]Ch;
for (i = 0; i<n; i++)
delete Ch1[i];
delete[]Ch1;
}
```