目 次

第1章　序論

QR コード [1] は，1994 年に株式会社デンソーウェーブが開発した二次元バーコード であり，食品や製造業の在庫管理など多方面の分野で利用されている．一般的な QR コードは，白と黒の正方形のモジュールで構成されておりデザイン性を考慮していない. 一方で広告，サービス業界ではデザイン性を考慮した QR コードが求められている．デザイン性を考慮した QR コードでは，一定のルールから QR コードを変更することによって，QR コード上にロゴ画像（以後，目的画像と述べる）を埋め込んだものがある．このような QR コードを Aesthetic QR コード [3] という．

Aesthetic QR コードの研究は大きく三種類に分けることができる：

1. QR コードの一部に目的画像を埋め込む方法,
2. 画像のヒストグラムを考慮して目的画像を埋め込む方法 [2],
3. QR コードに利用されている RS 符号中の padding codewords と呼ばれる領域 (以下，埋め草コード語) を考慮して目的画像を埋め込む方法 [3].

上に述べた 1，2 の方法についてはソフトウェアが公開されており，誰でも作成することができる．一方で，3 番目の手法は目的画像を QR コード全体に埋め込む手法として研究されているが，一般に公開されたソフトウェアは存在せず，誰もが利用できる状態にはなっていない．

本研究では，3 番目の手法に分類される目的画像に近い Aesthetic QR コードを自動生成するソフトウェアの開発について検討する．Aesthetic QR コードを自動生成する手法として，本研究では，Kuribayashi らの論文 [3] で提案されている Ramdom Method のアルゴリズムを用いる．QR コードで用いられる RS 符号の計算は代数拡大体上で計算されるため，本研究では数式処理システム Maple を用いて Aesthetic QR コードを生成するソフトウェアを開発した．

以下，第 2 章では QR コードを構成する Reed-Solomon 符号と QR コードの概要について述べ，第 3 章では本研究で用いる Kuribayashi らの論文 [3] の Ramdom Method，Color Translation

について述べる．第 4 章では数式処理 Maple による AesthiticQR コードの実装とその結果について述べる．第 5 章では結論と今後の課題について述べる．

第2章　QRコード

この章では AestheticQRcode をの構成要素である Reed-Solomon 符号（以下，RS 符号）と QR コードについて述べる．

2.1　QR コードの概要

QR コードの構成要素の最小単位は白と黒で表されるモジュールであり，白のモジュールは 0 のビット値を表し，黒のモジュールは 1 のビット値を表す．QR コードのサイズはバージョンによって決定され，そのバージョン (v) は 1 ~ 40 である．1 型は，21 × 21 モジュール，2 型は，25 × 25 モジュール，というように，型番が一つ上がるごとに一辺につき 4 モジュールずつ増加し，40 型は，177 × 177 モジュールとなる. したがって，バージョン v は $(17 + 4v) \times (17 + 4v)$ モジュールである．

図 2.1 に QR コードの構成要素を表す．

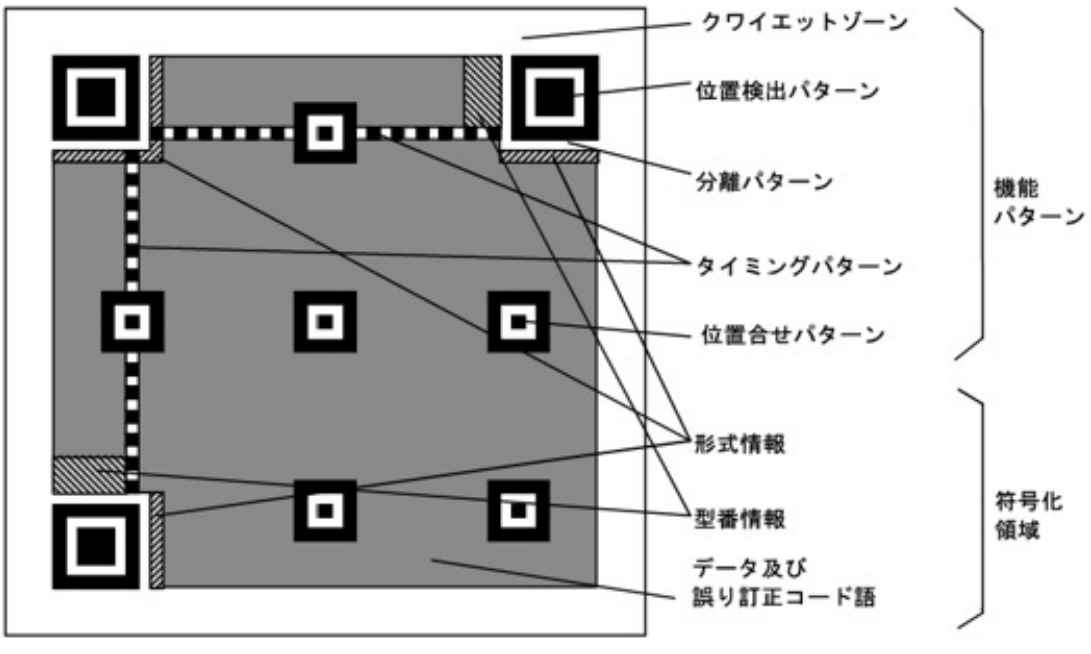

図 2.1　QR コードシンボルの構造 [8]

QR コードは，QR コード上にある符号化されたデータを正確に認識するために機能パターンを持っている．機能パターンは主に 3 つの構成要素から成り立っており，それぞれ位置検出パターン，位置合わせパターン，タイミングパターンと呼ばれる．

位置検出パターンは QR コードの左上，左下，右上の角にある 3 つの正方形のブロックである. それらの境目を明白にするために，形式情報との間に白いモジュールを置く. これを分離パターンと呼ぶ.

位置合わせパターンは小さな正方形のブロックで，位置検出パターンの垂直・平行座標に関係する位置に置く. バージョンによっては付加しない場合もあり，バージョン 1 には存在しない．

タイミングパターンは左上の位置検出パターンから右上の位置検出パターンへと，左上の位置検出パターンから左下の位置検出パターンへの白黒が交互に並ぶ 2 つのラインのことである．

QR コードのデータビットは，QR コードの右下から始まり，2 モジュール幅の列上に配置する．列が最上部に達すると，次の 2 モジュール列は右端から始まり，下方向へ続く．現在の列が端に達すると，次の 2 モジュールの列に移動して方向を変更する．データビットは機能パターン（位置検出パターン，タイミングパターン，位置合わせパターン）の位置では，次のモジュールへ配置される．

上方向のデータビットの配置は図 2.2 に，下方向のデータビットの配置は図 2.3 に示す．

0	1
2	3
4	5
6	7

図 2.2　上方向のビット配列

6	7
4	5
2	3
0	1

図 2.3　下方向のビット配列

またデータの格納方法にも種類があり，英数字モードや 8 ビットバイトモードなどがある．

QR コードは，誤り訂正符号として RS 符号を使用し，その能力は L，M，Q，H の 4 つのレベルに昇順で分類される．各誤り訂正レベルは QR コード内の全シンボルの約 7%，約 15%，約 25%，約 30% までのシンボルを訂正することができる．それぞれを表 2.1 に示す．

表 2.1　誤り訂正レベル

レベル	L	M	Q	H
誤り訂正能力	約 7%	約 15%	約 25%	約 30%

2.2　RS 符号

RS 符号とは符号理論における誤り訂正符号の一つである. その高い誤り訂正能力から，QR コードなどに応用されている．

以下に，RS 符号の各用語について説明する．

1. 符号多項式

   符号長 n の線形符号 C の任意の符号語をベクトル表現

   $$u = (u_0, u_1, u_2, \cdots, u_{n-1}) \tag{2.1}$$

   としたとき式 2.1 の多項式表現は

   $$u(x) = u_0 + u_1 x + u_2 x^2 + \cdots + u_{n-1} x^{n-1} \tag{2.2}$$

   である. ここで変数 x^i は単に記号 u_i の位置を示すだけである．式 2.2 のようにある符号語に対応する多項式を特に符号多項式と呼ぶ.

2. 生成多項式

   ある情報記号と対応する多項式 (以下，情報多項式) $q(x)$ からこれの誤り訂正を行う符号語 $u(x)$ を生成することを考えた際

   $$u(x) = q(x)g(x) \tag{2.3}$$

   と表される $g(x)$ を生成多項式と呼ぶ.

3. 体

   体とは代数学においてある性質を満たした集合である．体の性質の中でも最も特徴的な点は，元 (体の要素をこのように呼ぶ) の四則演算は結果も元になる (体の中で閉じている) という点である．例えば実数は体であり，実数を用いた四則演算は計算結果が全て実数になる. しかし自然数は体ではなく，例えば 1 – 2 の演算結果は負の値となりこれは自然数ではないのでこれは体とは言えない.

4. ガロア体 (有限体)

体の中でも元が有限なものをガロア体 (Galois field) と呼び，有限体とも呼ばれる．元の数が q のガロア体を $GF(q)$ で表し，元の数は素数，あるいは素数のべき乗である必要がある．例えば $GF(2)$ の元は一般的に 0 と 1 である. 計算例として以下に $GF(2)$ 上の加算減算の計算結果を示す.

表 2.2　GF(2) 上の加算結果

入力 1	入力 2	出力
0	0	0
0	1	1
1	0	1
1	1	0

表 2.3　GF(2) 上の減算結果

入力 1	入力 2	出力
0	0	0
0	1	1
1	0	1
1	1	0

5. 拡大体, 原始元, 原始多項式

ガロア体 $GF(p)$ 上の既約多項式 (これ以上因数分解できない多項式) $g(x)$ を選び，その根 ($g(x) = 0$ となるような x の値) を α とする．この α を $GF(p)$ の元に追加することで新たな体が生成でき，そうしてできた新たな体を拡大体と呼ぶ．この時の α を原始元といい，この既約多項式は原始多項式という．拡大体の例として複素数が挙げられる．複素数は実数の拡大体であり実数上の既約多項式 $x^2 + 1$ の根を i として実数の元に追加したものである．

拡大体 $GF(2^m)$ 上の RS 符号について考える．拡大体 $GF(2^m)$ の原始元を α とするとき，$\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t}$ を根として持つ $GF(2^m)$ 上の生成多項式 $g(x)$ により生成される符号を t 重誤り訂正 RS 符号と呼ぶ.

QR コード上の生成多項式 [5] [6] は $n - k$ 次多項式であり，これを $g(x)$ とする．

$$g(x) = (x-1)(x-\alpha)\cdots(x-\alpha^{n-k-1}) = \prod_{i=0}^{n-k-1}(x-\alpha^i) \tag{2.4}$$

式 2.4 の $g(x)$ を展開した多項式を

$$g(x) = g_1 x^{n-k} + g_2 x^{n-k-1} + \cdots + g_{n-k+1},\ g_1 = 1 \tag{2.5}$$

とする．その係数列 $g_1 = 1, g_2, \cdots, g_{n-k1}$ から定まる $k \times n$ 行列

$$G_0 = \begin{bmatrix} 1 & g_2 & \cdots & g_{n-k-1} & 0 & \cdots & \cdots & 0 \\ 0 & 1 & g_2 & \cdots & g_{n-k-1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k-1} & 0 \\ 0 & \cdots & \cdots & 0 & 1 & g_2 & \cdots & g_{n-k-1} \end{bmatrix}$$

を生成行列と呼ぶ.

一般的に QR コードでは情報記号と検査記号を分けた組織符号が用いられる．行列 G_0 を使って組織符号を計算するためには，行列 G_0 を”掃き出し法”によって $G = [I_k, P]$ (I_k は $k \times k$ 単位行列) という標準形に変形する．

データ $v = (v_1, v_2, \cdots, v_k)$ に対する符号語 u は標準形の生成行列 $G = [I_k, P]$ により，

$$u = vG \tag{2.6}$$

として表現することができ，この u が QR コード上の RS 符号である．

2.2.1　QR コード上の RS 符号

以下に QR コード上での RS 符号について示す.

RS 符号の各シンボルは拡大体 $GF(2^8)$ を使用し，原始多項式は $x^8 + x^4 + x^3 + x^2 + 1$ を使用する．この原始多項式の原始元 α を用いて計算を行う.

RS 符号の情報長は固定されているため，QR コードに入れる情報が少ない場合は埋め草コード語が追加される．例えばバージョン 1 の QR コードを構成する RS 符号の長さは $n = 26$ シンボルとなっているが，そのうちの情報記号の長さは 16 シンボルである.QR コードに入れる文字列が $k = 16$ シンボル未満で表現される場合，情報記号のシンボル数を 16 シンボルに合わせるため，埋め草コード語という情報を持たないシンボルを付加する必要がある．主に QR コードに書き込む文字列を RS 符号化したものを含むシンボル (以下，データと述べる) の個数を $\hat{k}$ と表すとき，データを表す RS 符号のシンボルは

$$\alpha_1, \cdots, \alpha_{\hat{k}} \tag{2.7}$$

である．$\hat{k} < k$ の時，RS 符号のデータを表すシンボルを k 個に合わせるため，埋め草コード語を付加する．埋め草コード語は

$$\alpha_{\hat{k}+1}, \cdots, \alpha_k \tag{2.8}$$

と表す.

これにより，RS 符号の情報記号のシンボルは

$$\alpha_1, \cdots, \alpha_{\hat{k}}, \alpha_{\hat{k}+1}, \cdots, \alpha_k$$

と表せる.

QR コード上の RS 符号の全体図を図 2.4 に示す.

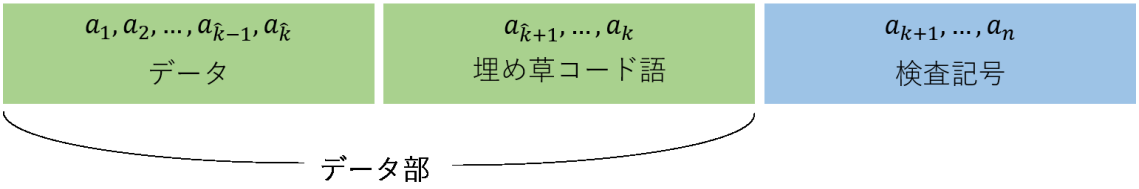

図 2.4　QR コード上の RS 符号の全体図

第3章　Aesthetic QRコード

本章では，Kuribayashi らの論文で提案された [3] のランダム法 (Random-Method), 色変換手法 (Color Translation) について述べる．ランダム法は QR コード上に配置された二値配列が埋め込む目的画像のモジュールパターンと類似した Aesthetic QR コードを生成するアルゴリズムである．色変換手法はカラー画像に対する Aesthetic QR コードを生成するためのアルゴリズムである．Aesthetic QR コードの例を図 3.1 に示す．このように Aesthetic QR コードはデザイン性が考慮されている．

図 3.1　Asethetic QR コード例 [3]

3.1　目的画像の二値化手法

目的画像はカラー画像でも二値画像でもよい．カラー画像や二値画像を用いる際に，目的画像と QR コードとの距離を測るために，目的画像を二値化する必要がある．ここでの距離にはハミング距離を用い，最もハミング距離の小さい QR コードを用いて Aesthetic QR コードを作る．目的画像を二値化画像にする際に閾値を指定するが，その閾値を Algorithm 1 に示す閾値変換手法で決定する．

閾値を用いて目的画像から目的画像の二値行列を作成するアルゴリズムを Algorithm 2 に示す．

Algorithm 1 論文 [3] の閾値計算手法

入力: サイズ $L \times L$ の目的画像

出力: 閾値 $\overline{Y}$

方法:
1. 入力画像の大きさは，QR コードのバージョン v と同じサイズに予め変更しておく．RGB 色成分は YUV 色成分に変換され，輝度 (Y) 成分 $Y_{i,j}$,$(1 \leq i, j \leq L)$ が得られる．
2. その中心の正方形 (元の画像サイズの $\frac{1}{4}$) の値の平均 $\overline{Y}$ が計算する．

$$\overline{Y} = \frac{4}{L^2} \sum_{i=\frac{L}{4}}^{\frac{3L}{4}-1} \sum_{j=\frac{L}{4}}^{\frac{3L}{4}-1} Y_{i,j} \tag{3.1}$$

Algorithm 2 論文 [3] の目的画像に対する二値行列の生成

入力: サイズ $L \times L$ の目的画像の輝度 (Y) 成分 $Y_{i,j}(1 \leq i, j \leq L)$，閾値 $\overline{Y}$

出力: 二値行列 $B_{i,j}$

方法:
1. 目的画像を二値化する際，二値行列 $B_{i,j}$ は，以下の規則によって決定される．

$$B_{i,j} = \begin{cases} 1 & Y_{i,j} > \overline{Y} \\ 0 & otherwise \end{cases} \tag{3.2}$$

3.2　ランダム法

ランダム法とは QR コードのシンボルの順番を規則に従って変えることで，目的画像を二値化した画像との距離が小さい QR コードを生成する方法である. まず，AestheticQR コードの元となる QR コードを生成するための生成行列 G を求める．Random Method ではこの G を用いて，掃き出し法により QR コードを生成する. ただし，単位行列の列ベクトルの位置は情報記号の最初の $\hat{k}$ 列を除いてランダムに決定される．次に，QR コードと目的画像の画像サイズを等しいものとして，QR コードの 1 モジュールと目的画像の 1 画素を対応させ，目的画像と QR コードがどれだけ近いものか比較する．ただし，QR コードに書き込むデータのシンボルを除いて比較する．具体的には，目的画像の二値化を行ったものに QR コードのデータ (RS 符号中の情報記号の中のデータにあたる箇所) を書き加えたものを用意し，それと生成した QR コードとのハミング距離を取る．以上を N 回繰り返しハミング距離が最小となる QR コードを求め，それを使って Aesthetic QR コードを作成する．

本研究では QR コードの中でバージョン 1 の QR コードを用いた．バージョン 1 の QR コードを作成する手順を Algorithm3 に示す.

Algorithm 3 論文 [3] のランダム法を用いたバージョン 1 の Aesthetic QR コード

入力 (試行回数 N): バージョン 1 の QR コードに入るシンボル長 16 の文字列，サイズ 21×21 の目的画像

出力: サイズ 21×21 のバージョン 1 の Aesthetic QR コード

方法:

1. 目的画像の画素値を QR コードのモジュールに割り当て，Algorithm2 で決まった閾値 $\overline{Y}$ でモジュールを二値化し，二値行列 $B_{i,j}$ を作成する．
2. $B_{i,j}$ に所定のマスキングパターンを作用させる．
3. 式 (2.8) の α_t の位置 t ($\hat{k}+1 \leq t \leq n$) を変化させることにより，RS 符号を計算する．以後この手順を N 回繰り返すことにより，$B_{i,j}$ とのハミング距離が最小となる RS 符号を見つける．一定の試行回数終了後，QR コード上の RS 符号は，ハミング距離が最小の RS 符号に置き換える．
4. マスク処理前である QR コードの各モジュールに対して，所定のマスクパターンを適用し，AestheticQR コードとして出力する．

3.3　色変換手法

色変換手法とは，Algorithm3 までで生成した Aesthetic QR コードに色を追加するアルゴリズムである．例えば，Algorithm3 までで生成した Aesthetic QR コード内の黒の画素があったとし，この画素をそのまま同じ位置にある目的画像の画素の色に変換してしまうと，目的画像の輝度値によっては本来 1 と読み取るはずの画素が 0 と判別されてしまうなどして，Aesthetic QR コードとして読み込まれないという問題が生じる可能性がある．そのため，そのまま目的画像の色に変換するのではなく，画素ごとに輝度値を変更する必要がある．例えば，Aesthetic QR コードとして 1 と読み取る画素は輝度値を変更してある程度暗く，反対に 0 と読み取る画素はある程度明るくする必要がある．このアルゴリズムはその輝度値を変更するためのアルゴリズムである．

具体的にはまず，目的画像の平均輝度値を求めこれを $\overline{Y}$ とする．次に閾値 ϵ(本実験では実験的に 0.25 とする) を定める．ϵ とはある画素を平均輝度値からどれだけ離れた値に変更するかという値である．

色変換手法を適用するアルゴリズムを以下に示す．

Algorithm 4 論文 [3] の色変換手法

入力: algorithm3 で生成した Aesthetic QR コード

入力: 目的画像

出力: カラー画像に対する Aesthetic QR コード

方法:
1. 目的画像を入力の Aesthetic QR コードの大きさに変更する，
2. 大きさを変更した目的画像の平均輝度値を求める，
3. 大きさを変更した目的画像の画素 $\beta_{i,j}$ の輝度値 $Y_{i,j}$ を，以下の方法にしたがって新しい輝度値 $Y'_{i,j}$ に変更する．

   if $\beta_{i,j} = 1$,then

$$Y'_{i,j} = \begin{cases} Y_{i,j} & Y_{i,j} > \overline{Y} + \epsilon \\ \overline{Y} + \epsilon & otherwise \end{cases} \tag{3.3}$$

   otherwise

$$Y'_{i,j} = \begin{cases} Y_{i,j} & Y_{i,j} < \overline{Y} - \epsilon \\ \overline{Y} - \epsilon & otherwise \end{cases} \tag{3.4}$$

第4章　数式処理システムMapleを用いたAestheticQRコードの実装

4.1　数式処理システム Maple [7]

Maple は数式を正確に誤差なく計算するためのシステムである．数式の計算には記号計算や数値計算，グラフ描画などが含まれる．Maple は 1985 年にカナダのウォータールー大学で開発が始められた．現在，世界中で使用されており，科学技術計算や工学問題や教育などに応用されている．主に計算可能な数式としては以下のものがあげられる．

- 多倍長整数演算
- 多項式演算
- 行列ベクトル演算
- 代数体上での計算 (ガロア体の計算)
- 数値計算

また，Maple で定義されたプログラミング言語があり，その言語を使って新しい数学関数を定義したり，ユーザーインターフェースを作成したりすることができる．

4.2　実験環境

実験に使用した PC 環境，言語を以下に示す．

- ソフトウェア実装環境
  - CPU：Intel(R)Core i5 7500 3.4GHz
  - OS：Windows 10 pro
  - 実装 RAM：16.0GB

- 開発環境
  - Maple：Maple 2021.1

実験に使用した各パラメータを以下に示す.

- QR コード
  - RS 符号：(26,16) 符号
  - 入力文字：tahara
  - バージョン (v)：1
  - マスクパターン：001
  - 誤り訂正レベル：M

QR コードのバージョン 1 の構造を図 4.1 に示す.

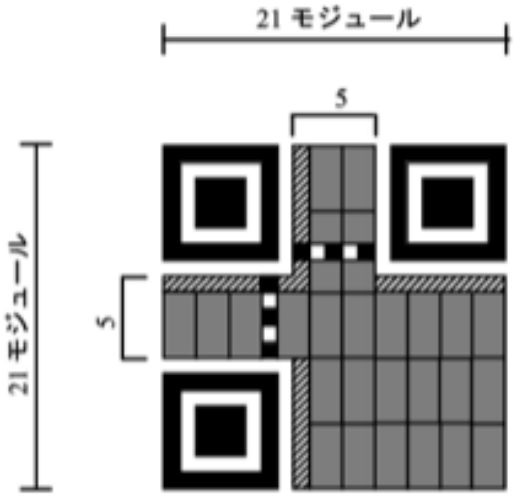

図 4.1　QR コードのバージョン 1 の構造

4.3　Maple による QR コードの実装

本研究では，RS 符号を生成する際に必要な有限体の実装を行うために，Aesthetic QR コードの生成に数式処理 Maple(以下，Maple) を用いた．以下に Maple において，QR コード中の RS 符号を生成する方法を示す.

拡大体 $GF(2^8)$，原始多項式 $x^8+x^4+x^3+x^2+1$ とその原始元 α は以下のように表される．Maple において原始多項式は GF 関数の第 3 引数で定義される．GF はガロア体を定義するための関数である.

```
G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1):
a := G8:-ConvertIn(alpha):
```

図 4.2　Maple 上でのガロア体の実装

Algorithm1～3 の実装を Maple 上で行い，ソースコードを付録 A に示す．Aesthetic QR コードを生成するための関数は gen_AestheticQRcode である．gen_AestheticQRcode は，Algorithm2 で生成した QR コードと，目的画像を入力として，Aesthetic QR コードを生成する．結果を図 4.4 に示す．

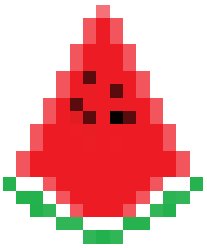

図 4.3　目的画像

```
> #関数名：gen_AestheticQRcode
  #入力１：QRコードの情報が入ったベクトル
  #入力２：目的画像を示すパス
  #出力：AestheticQRcodeの情報が格納された行列

  AestheticQRcode := gen_AestheticQRcode(AestheticQRcode_Vec,image_path):
  Preview(AestheticQRcode);
```

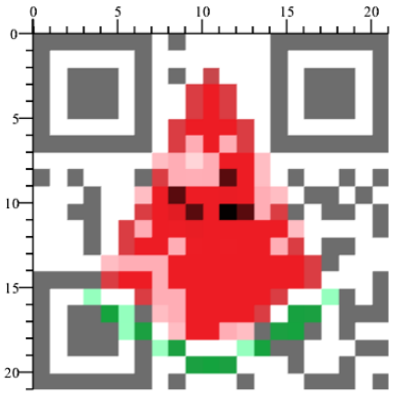

図 4.4　関数 AestheticQRcode の入力と出力

4.4　Aesthetic QR コードの生成に関する実験

Aesthetic QR コードは以下の 2 通りの方法で行う．

- 目的画像の二値化行列 $B_{i,j}$ と Algorithm2 で得られた QR コードの二値表列のハミング距離
- 一つの Aesthetic QR コードの生成に要した時間

ハミング距離は Kuribayashi らの論文 [3] で使われている．

2 つの符号語つの符号語 $a = (a_1, a_2, \cdots, a_n)$ と $b = (b_1, b_2, \cdots, b_n)$ で対応するビット (桁）で値 (0 または 1) が異なっているビット (桁) の数をハミング距離と言い，記号で $d(a, b)$ と書く．その中でも一番ハミング距離が小さいものを最小ハミング距離と呼ぶ．

ハミング距離は 2 つの符号語 $a = (a_1, a_2, \cdots, a_n)$ と $b = (b_1, b_2, \cdots, b_n)$ に対して以下の式で定義される．

$$d(a, b) = \sum_{i=1}^{n} (a_i + b_i) \pmod 2 \tag{4.1}$$

本実験では QR コードに挿入する画像は以下の目的画像 (図 4.5) を使用した．図 4.6~4.10 はランダム法の試行回数が $N = 1, 10, 100, 1000, 10000$ の場合の結果 (Aestheic QR コード) を表す．実際にソフトウェアによって生成した画像が図 4.6~ 図 4.10 である．

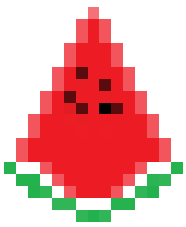

図 4.5　目的画像

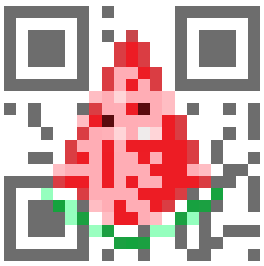

図 4.6　$N = 1$

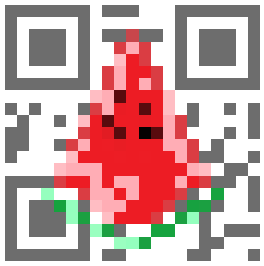

図 4.7　$N = 10$

図 4.8　$N = 100$

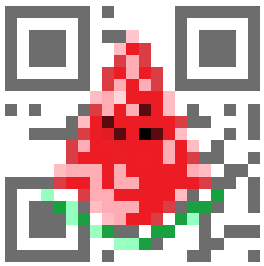

図 4.9　$N = 1000$

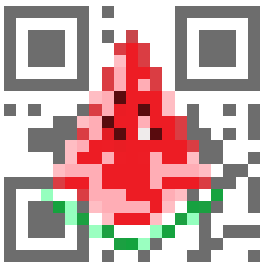

図 4.10　$N = 10000$

また，表 4.1 に各試行回数における実験で得られたハミング距離，RS 符号の生成にかかった時間を表す.

表 4.1　結果のまとめ

生成した回数 (N)	1	10	100	1000	10000
最小ハミング距離	83	63	51	47	41
時間（秒）	0.08	0.81	8.01	80.39	771.17

表 4.1 を見ると，試行回数を増やしていくたびに評価の値が向上しており，より良い Aesthetic QR コードが生成されているのがわかる．しかし，試行回数 10000 回の所を見ると RS 符号が生成するのにかかる時間が約 13 分もかかっており，Maple 上の実装について計算時間の短縮が検討課題である．

第5章　結論

本研究では，数式処理 Maple 上での Aesthetic QR コードを自動生成するソフトウェアを開発した．目的画像に近い Aesthetic QR コードを自動生成する方法について，Kuribayashi らの手法を数式処理システム Maple 上に実装した．その結果，任意の目的画像に対する AestheticQR コードを生成することができ，さらに Kuribayashi らの手法の有効性も確認することができた．しかし，今後の課題として

- 生成速度を短縮するアルゴリズムの検討
- 生成可能な Aesthetic QR コードのバージョンの追加

などが今後の課題として挙げられる．

謝辞

本研究に際して，日々，様々なご指導をいただきました甲斐博准教授に心より感謝致します．最後に日頃から助言や励ましをいただきました諸先輩方，並びに同研究室の皆様に深く御礼を申し上げます．

参考文献

［1］ QR code.com. http://www.qrcode.com/en.

［2］ Visualead. http://www.visualead.com.

［3］ M. Kuribayashi and M . Morii ”Aesthetic QR Code Based on Modified Systematic Encoding Function”, IEICE transactions on information and systems ,VOL.E100–D, NO.1,pp.42-51,2017.

［4］ 汐崎陽，情報 · 符号理論の基礎, 2011 年

［5］ 池田和興, 例題が語る符号理論, 共立出版, 2007 年

［6］ J. Justesen and T. Hoholdt ”A Course In Error-Correctiong Codes”, European Mathematical Society Publishing House, 2004.

［7］ B. W. Char, K. O. Geddes, G. H. Gonnet, B. L. Leong, M. B. Monagan, and S. M. Watt, First Leaves: A Tutorial Introduction to Maple V, Springer-Verlag, 1992.

［8］ JIS X0510. 情報技術－自動認識及びデータ取得技術－ＱＲコード　バーコードシンボル体系仕様
http://www.jisc.go.jp/app/pager?id=2738494.

付 録A　プログラムリスト

ソースコード A.1　A_QRcode_Naoya

```
restart;
isFinderPattern := proc(version, x, y)
 local size;
 size := QRcodeSize(version);
 return x <= 7 and y <= 7 or x <= 7 and size - 8 <= y or size - 8 <= x and y <= 7;
end proc;

isTimingPattern := proc(version, x, y)
 return not isFinderPattern(version, x, y) and (x = 6 or y = 6);
end proc;

isFunctionPattern := proc(version, x, y)
 return isFinderPattern(version, x, y) or isTimingPattern(version, x, y);
end proc;

dataWritable := proc(version, x, y)
 local size;
 size := QRcodeSize(version);
 if isFunctionPattern(version, x, y) then
  return false;
 end if;
 if x <= 8 and y <= 8 then
  return false;
 end if;
 if x = 8 and size - 8 <= y then
  return false;
 end if;
 if y = 8 and size - 8 <= x then
  return false;
 end if;
 if 7 <= version then
   if x < 6 and size - 11 <= y and y < size - 8 then
  return false;
  end if;
  if y < 6 and size - 11 <= x and x < size - 8 then
   return false;
  end if;
 end if;
 if x = 8 and y = 4*version + 9 then
  return false;
```

```
 end if;
 return true;
end proc;

PosProceed := proc(version, xref::uneval, yref::uneval)
 local size, x, y, relx, rely; x := eval(xref); y := eval(yref);
 size := QRcodeSize(version);
 ASSERT(x <> 6);
 relx := x; rely := y;
 if 6 < x then
  relx := relx - 1;
 end if;
 if relx mod 2 = 0 then
  if iquo(relx, 2) mod 2 = 0 then
   if rely = size - 1 then
    if relx = 0 then
     x := size - 1;
    else x := x - 1;
   end if;
  else x := x + 1; y := y + 1;
 end if;
  else if rely = 0 then
   if x = 7 then
    x := x - 2;
  else x := x - 1;
  end if;
  else x := x + 1;
  y := y - 1;
  end if;
 end if;
 else x := x - 1;
end if;
xref := x; yref := y;
end proc;

PosNext := proc(version, x_::uneval, y_::uneval)
 local size, x, y; x := eval(x_); y := eval(y_);
 size := QRcodeSize(version);
 do
  break;
  if x = 0 and y = size - 1;
  PosProceed(version, x, y);
  break;
  if dataWritable(version, x, y);
 end do;
 x_ := x; y_ := y;
end proc;

setVec := proc(x, y, exp, in_Vec, version)
 local Vec, M; M := QRcodeSize(version);
 Vec := in_Vec;
```

```
 if evalb(exp) then
  Vec[M*y + x + 1] := "Black";
 else
  Vec[M*y + x + 1] := "White";
 end if;
end proc;

fill := proc(centerX, centerY, halfWidth, exp, Vec, version)
 local x, y, size;
 size := QRcodeSize(version);
 for y from max(centerY - halfWidth, 0) to min(centerY + halfWidth, size - 1) do
  for x from max(centerX - halfWidth, 0) to min(centerX + halfWidth, size - 1) do
   setVec(x, y, exp, Vec, version);
  end do;
 end do;
end proc;

PlaceFinderPattern := proc(Vec,version)
  local size;
  #global QRcodeVersion;
  size := QRcodeSize(version):

  #左 上
  fill(3, 3, 4, false,Vec,version):
  fill(3, 3, 3, true,Vec,version):
  fill(3, 3, 2, false,Vec,version):
  fill(3, 3, 1, true,Vec,version):

  #右 上
  fill(size - 1 - 3, 3, 4, false,Vec,version):
  fill(size - 1 - 3, 3, 3, true,Vec,version):
  fill(size - 1 - 3, 3, 2, false,Vec,version):
  fill(size - 1 - 3, 3, 1, true,Vec,version):

  #左 下
  fill(3, size - 1 - 3, 4, false,Vec,version):
  fill(3, size - 1 - 3, 3, true,Vec,version):
  fill(3, size - 1 - 3, 2, false,Vec,version):
  fill(3, size - 1 - 3, 1, true,Vec,version):

end proc:

PlaceTimingPattern := proc(Vec, version)
 local i, size;
 size := QRcodeSize(version);
 for i from 8 to size - 9 do
  setVec(i, 6, i mod 2 = 0, Vec, version);
 end do;
 for i from 8 to size - 9 do
  setVec(6, i, i mod 2 = 0, Vec, version);
 end do;
```

```
143 end proc;
144
145 PlaceAlwaysBlack := proc(in_Vec, version)
146  local size, Vec;
147  Vec := in_Vec; size := QRcodeSize(version);
148  Vec[size*(size - 8) + 9] := "Black";
149 end proc;
150
151 PlaceFunctionPattern := proc(Vec, version)
152  PlaceTimingPattern(Vec, version);
153  PlaceFinderPattern(Vec, version);
154  PlaceAlwaysBlack(Vec, version);
155 end proc;
156
157 PlaceFormatInfo := proc(Vec, version)
158  local Gxx, formatInfox, errCorCodex, result, maskPattern, deg, pos_x, pos_y, size, M, x
         ; x := 'x'; Gxx := x^10 + x^8 + x^5 + x^4 + x^2 + x + 1;
159  formatInfox := 1;
160  errCorCodex := formatInfox*x^10;
161  errCorCodex := rem(errCorCodex, Gxx, x) mod 2;
162  result := formatInfox*x^10 + errCorCodex;
163  maskPattern := x^14 + x^12 + x^10 + x^4 + x;
164  result := (result + maskPattern) mod 2;
165  deg := 14;
166  for pos_x from 0 to 5 do setVec(pos_x, 8, coeff(result, x, deg) = 1, Vec, version);
167  deg := deg - 1;
168  end do;
169  setVec(7, 8, coeff(result, x, deg) = 1, Vec, version);
170  deg := deg - 1;
171  setVec(8, 8, coeff(result, x, deg) = 1, Vec, version);
172  deg := deg - 1;
173  setVec(8, 7, coeff(result, x, deg) = 1, Vec, version);
174  deg := deg - 1;
175  for pos_y from 5 by -1 to 0 do
176   setVec(8, pos_y, coeff(result, x, deg) = 1, Vec, version);
177  deg := deg - 1;
178  end do; deg := 14;
179  size := QRcodeSize(version);
180  for pos_y from size - 1 by -1 to size - 7 do
181   setVec(8, pos_y, coeff(result, x, deg) = 1, Vec, version);
182  deg := deg - 1;
183  end do;
184  for pos_x from size - 8 to size - 1 do
185   setVec(pos_x, 8, coeff(result, x, deg) = 1, Vec, version);
186  deg := deg - 1; end do;
187 end proc;
188
189 flip := proc(x, y, in_Vec, version) local size, Vec;
190  Vec := in_Vec;
191  size := QRcodeSize(version);
192  if Vec[x + 1 + size*y] = "White" then
```

```
193   Vec[x + 1 + size*y] := "Black";
194  elif Vec[x + 1 + size*y] = "Black" then
195   Vec[x + 1 + size*y] := "White";
196  end if;
197 end proc;
198
199 myMask := proc(maskPattern, in_Vec, version) local condition, size, x, y, Vec;
200  size := QRcodeSize(version);
201  for y from 0 to size - 1 do
202   for x from 0 to size - 1 do
203    condition := false;
204    if maskPattern = "000" then
205     condition := evalb((x + y) mod 2 = 0);
206    elif maskPattern = "001" then
207     condition := evalb(y mod 2 = 0);
208    elif maskPattern = "010" then
209     condition := evalb(x mod 3 = 0);
210    elif maskPattern = "011" then
211     condition := evalb((x + y) mod 3 = 0);
212    elif maskPattern = "100" then
213     condition := evalb((iquo(x, 3) + iquo(y, 2)) mod 2 = 0);
214    elif maskPattern = "101" then
215     condition := evalb(((x*y mod 3) + x*y) mod 2 = 0);
216    elif maskPattern = "110" then
217     condition := evalb((((x*y mod 3) + x*y) mod 2) mod 2 = 0);
218    elif maskPattern = "111" then
219     condition := evalb((((x*y mod 3) + x + y) mod 2) mod 2 = 0);
220    else print("マスクパターン値が異常です");
221     condition := false;
222    end if;
223    Vec := in_Vec;
224    if condition then
225     flip(x, y, Vec, version);
226    end if;
227   end do;
228  end do; end proc;
229
230 SetModule := proc(x, y, isBlack, in_Vec)
231  local Vec;
232  Vec := in_Vec;
233  if isBlack then Vec[x + M*y] := "Black";
234   else Vec[x + M*y] := "White";
235  end if;
236 end proc;
237
238 PlaceCode := proc(codePairs, Vec, version)
239  local x, y, index, i, size;
240  global QRcodeVersion;
241  size := QRcodeSize(version);
242  x := size - 1; y := size - 1;
243  for index to numelems(codePairs) do
```

```
  for i to 8 do setVec(x, y, codePairs[index][i] = 1, Vec, version);
   PosNext(QRcodeVersion, x, y);
  end do;
 end do;
end proc;

#関数：gen_QRcode
#出力：QRcodeの内容が入ったVec

gen_QRcode := proc(str,L,image_path,n,k,khat,QRcodeVersion,For_QRcode_binary_image)
local i,j;
local delta,myrand,myset,mylist;

#deltaを求める
delta := Vector(k):
for i from 1 to k-khat do
  delta[i] := i;
end do:
randomize():
myrand := rand(k-khat+1..n):
myset := {}:
while (nops(myset)<>khat) do
  myset := `union`(myset, {myrand()});
end do:
mylist := convert(myset, list);

#mylist:=[13,14,15,16];#####Debug
#mylist:=[9, 10, 14, 15, 19, 22, 24, 26];#####Debug

for i from 1 to khat do
  delta[i+k-khat] := mylist[i];
end do:

#Gの生成
local inv,l,tmp,G;
G := Matrix(k,n):
for i from 1 to k do
  for j from 1 to n do
    G[i, j] := G8:-input(0);
  end do;
end do;
for i from 1 to k do
  for j from 1 to 11 do
    G[i,(i-1)+j] := G8:-`^`(a,GP[j]);
  end do;
end do;
for i from 1 to k do
  inv := G8:-inverse(G[i,delta[i]]):
  for j from 1 to n do
    G[i,j] := G8:-`*`(inv, G[i,j]);
  end do;
```

```
  for l from 1 to i-1 do
    tmp := G[l,delta[i]];
    for j from 1 to n do
      G[l,j] := G8:-`-`(G[l,j], G8:-`*`(tmp, G[i,j]));
    end do;
  end do;
  for l from i+1 to k do
    tmp := G[l,delta[i]];
    for j from 1 to n do
      G[l,j] := G8:-`-`(G[l,j], G8:-`*`(tmp, G[i,j]));
    end do;
  end do;
end do:


#FP_binの生成
local p,cnt,padding_location,code_num,size,x,y,Str_bin,FP_bin;
FP_bin := [0,1,0,0]:
FP_bin := [op(FP_bin),op(binarize(L))]:
Str_bin := Use_ToByteArray(str):
for i from 1 by 1 to L do
  tmp := op(binarize(Str_bin[i])):
  FP_bin := [op(FP_bin),tmp]:
end do:
FP_bin := [op(FP_bin),op([0,0,0,0])]:
#code_num：何コード目かを示す
#deltaのうち埋め草コード部のみをpadding_locationに格納
padding_location := [];
for p from k-khat+1 by 1 to n do
  if member(p,delta) then
    padding_location := [op(padding_location),p];
  end if;
end do;

size := QRcodeSize(QRcodeVersion);
x := size-1;
y := size-1;

cnt:=1;
code_num := floor((cnt-1)/8) + 1;
while code_num < n+1 do
  if code_num > k-khat and member(code_num, padding_location) then
    FP_bin := [op(FP_bin),floor(For_QRcode_binary_image[y+1,x+1]+1)mod 2];
  end if;
cnt++;
PosNext(QRcodeVersion,x,y);
code_num := floor((cnt-1)/8) + 1;
end do;
FP_bin := Use_LengthSplit(FP_bin,8):

#FP_listを作成する
```

```
local FP_list,For_FP_list,dec;
FP_list := []:
for i from 1 by 1 to k do
  For_FP_list := Use_Reverse(FP_bin[i]):
  dec := 0:
  for j from 1 by 1 to 8 do
    dec := dec + For_FP_list[j] * (2 ^ (j-1));
  end do:
  FP_list := [op(FP_list),get_exp(dec)]:
end do:

#FP,Fの生成
local FP,F;
FP := Vector(k, FP_list):
F := Vector(k):
for i from 1 to k do
  if (FP[i]>=0) then
    F[i] := G8:-`^`(a, FP[i]);
  else
    F[i] := G8:-input(0);
  end if;
end do;

#Cの生成
local C;
C := Vector(n);
for i from 1 to n do
  C[i] := G8:-input(0);
  for j from 1 to k do
    C[i] := G8:-`+`(C[i], G8:-`*`(F[j], G[j, i]));
  end do;
end do:

#Vecの生成
local maskPattern,Vec,M:
local c;
c := [];
for i to n do
    c := [op(c), mybin(G8:-output(C[i]))];
end do;

M := QRcodeSize(QRcodeVersion);
maskPattern := "001":
Vec := Vector(M*M):
for i to M*M do
    Vec[i] := "Pink";
end do:

PlaceCode(c,Vec,QRcodeVersion):
myMask(maskPattern,Vec,QRcodeVersion):
PlaceFormatInfo(Vec,QRcodeVersion):
```

```
PlaceFunctionPattern(Vec,QRcodeVersion):

return Vec:

end proc:
#関数名：Calculate_Hamming_distance
#入力１：目的画像を二値化し、文字列を加えて書き換えたもの
#入力２：生成したAestheticQRcodeの元となる情報(Vec)
#出力：二つのハミング距離

Calculate_Hamming_distance := proc(For_Hamming_binary_image,QRcode_Vec,k,khat);
local i,j,Hamming_distance,cnt,code_num,QRcode_Size;

Hamming_distance := 0;
cnt:=1:
code_num := floor((cnt-1)/8) + 1:

QRcode_Size := sqrt(numelems(QRcode_Vec));
i := QRcode_Size-1:
j := QRcode_Size-1:
#while code_num < k-khat+1 do
while code_num < n+1 do
  if code_num > k-khat then
    if QRcode_Vec[j*21+(i+1)] = "Black" then
      if For_Hamming_binary_image[i+1,j+1] = 1 then
        Hamming_distance++;
      end if;
    elif QRcode_Vec[j*21+(i+1)] = "White" then
      if For_Hamming_binary_image[i+1,j+1] = 0 then
        Hamming_distance++;
      end if;
    end if;
  end if;
#print(j*21+(i+1),i+1,j+1);
  cnt++:
  PosNext(1,i,j):
  code_num := floor((cnt-1)/8) + 1:
end do:

return Hamming_distance:
end proc:
read "//wfs01/Users/e1848taha/Desktop/thesis/Maple/A_QRcode_Naoya_Function.mpl";
str := "Tahara":
#image_path := "./pic/Lenna.jpg":
#image_path := "./pic/mican.png":
#image_path := "./pic/flower.png":
image_path := "./pic/suika.png":
n:=26: k:=16: QRcodeVersion:=1:
with(StringTools):
L := Length(str):
khat := 16 - (1 + 1 + L):
```

```
448
449 G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1):
450 a := G8:-ConvertIn(alpha):
451 GP := Vector(n-k+1,[0,251,67,46,61,118,70,64,94,32,45]):
452
453 FP_bin := [0,1,0,0]:
454 FP_bin := [op(FP_bin),op(binarize(L))]:
455 Str_bin := ToByteArray(str):
456 for i from 1 by 1 to L do
457   tmp := op(binarize(Str_bin[i])):
458   FP_bin := [op(FP_bin),tmp]:
459 end do:
460 FP_bin := [op(FP_bin),op([0,0,0,0])]:
461 For_Hamming_binary_image := gen_resized_binary_image(image_path):
462 size := QRcodeSize(QRcodeVersion):
463 x := size-1:
464 y := size-1:
465
466 cnt:=1:
467 code_num := floor((cnt-1)/8) + 1:
468
469 while code_num < k-khat+1 do
470   For_Hamming_binary_image[y+1,x+1] := ((FP_bin[cnt] + 1) mod 2):
471   cnt++:
472   PosNext(QRcodeVersion,x,y):
473   code_num := floor((cnt-1)/8) + 1:
474 end do:
475
476 For_QRcode_binary_image := Array(For_Hamming_binary_image):
477
478 #目的画像にマスク処理
479 #マスクは001
480 for i from 1 by 1 to 21 do
481   for j from 1 by 1 to 21 do
482     if i+1 mod 2 = 0 then
483       For_QRcode_binary_image[i,j] := (floor(For_QRcode_binary_image[i,j]) + 1) mod 2 ;
484     end if;
485   end do;
486 end do;
487 with(ColorTools):
488 #Preview(Read(image_path));
489 #Preview(For_Hamming_binary_image);
490 #Preview(For_QRcode_binary_image);
491 N := 1:
492
493 sum_time := 0:
494
495 #for CNT from 1 by 1 to 10 do
496 min_Hamming_distance := 21*21:
497 st := time();
498
```

```
 for i from 1 by 1 to N do
  QRcode_Vec := gen_QRcode(str,L,image_path,n,k,khat,QRcodeVersion,
      For_QRcode_binary_image):
  result_Hamming_distance := Calculate_Hamming_distance(For_Hamming_binary_image,
      QRcode_Vec,k,khat):
  if min_Hamming_distance > result_Hamming_distance then
    AestheticQRcode_Vec := QRcode_Vec;
    min_Hamming_distance := result_Hamming_distance;
  end if:
  #if N>100 then
  #  if (i mod (N/10))=0 then
  #    print((i/(N/100)),"%",min_Hamming_distance);
  #  end if;
  #end if;
#print(result_Hamming_distance);
 end do:
fin := time()-st:

#sum_time := sum_time + fin;
#print(fin);
#end do:

#avarage_time := sum_time/10;

print("time",fin);
print(min_Hamming_distance);
Preview(gen_AestheticQRcode(AestheticQRcode_Vec,image_path));
Write(write_path,gen_AestheticQRcode(AestheticQRcode_Vec,image_path)):
```

ソースコード A.2　A_QRcode_Naoya_Function

```
with(ImageTools):
QRcodeSize := proc(QRcodeVersion)
 return QRcodeVersion*4+17:
end proc:
#関数：cal_avarage_luminance
#入力1：平均輝度値を取得したいグレースケール画像(正方形)
#出力：入力1の平均輝度値

cal_avarage_luminance := proc(gray_image);
local i,j,L,s,g,avarage_luminance;

L := Height(gray_image);
s := round(L/4);
g := round(L*3/4);

avarage_luminance := 0;
for i from s by 1 to g do
  for j from s by 1 to g do
    avarage_luminance := avarage_luminance + gray_image[j][i];
  end do;
```

```
21 end do;
22 avarage_luminance := avarage_luminance/((g - s)^2);
23
24 return avarage_luminance;
25 end proc:
26 #関数：gen_resized_image
27 #入力1：元画像(正方形)
28 #入力2：サイズ変更後の画像の縦横の長さ
29 #出力：サイズを変更した画像
30
31 gen_resized_image := proc(image,L);
32 local resized_L,resized_image;
33
34 resized_L := L/Width(image);
35 resized_image := Scale(image,resized_L);
36
37 return resized_image;
38 end proc:
39 #関数：gen_resized_binary_image
40 #入力：画像のパス
41 #出力：元画像を21*21に縮小し、グレイスケールにしてから二値化した画像
42
43 gen_resized_binary_image := proc(image_path);
44 local i,j,image,resized_gray_image,avarage_luminance,resized_binary_image;
45
46 image := Read(image_path);
47 resized_gray_image := RGBtoGray(gen_resized_image(image,21));
48 avarage_luminance := cal_avarage_luminance(resized_gray_image);
49
50 resized_binary_image := Create(21,21,1);
51 for i from 1 by 1 to 21 do
52   for j from 1 by 1 to 21 do
53     if (resized_gray_image[i,j] >= avarage_luminance) then
54       resized_binary_image[i,j] := 1;
55     end if;
56   end do;
57 end do;
58
59 return resized_binary_image;
60 end proc:
61 with(StringTools):
62 Use_ToByteArray := proc(str):
63 return ToByteArray(str):
64 end proc:
65 with(ListTools):
66 #関数名：binarize
67 #入力1：10進数
68 #出力：入力を2進数にし、8ビット化したリスト
69
70 binarize := proc(dec):
71 local bin, i;
```

```
bin := Reverse(convert(dec,base,2)):
for i from 1 by 1 to 8-nops(bin) do
  bin := [0,op(bin)];
end do:
end proc:
Use_LengthSplit := proc(list,n):
return LengthSplit(list,n):
end proc:
Use_Reverse := proc(list):
return Reverse(list):
end proc:
mybin := proc(a)
  local i, t, r;
  t := a;
  r := [-1, -1, -1, -1, -1, -1, -1, -1];
  for i from 1 to 8 do
    r[i] := t mod 2;
    t := iquo(t, 2);
  end do;
  r := Reverse(r):
  return r;
end proc:



#関数名：get_exp
#入力1：2の8乗のガロア体上の多項式表現
#出力：多項式表現の元となるアルファの冪

get_exp := proc(dec):
local exp, fin, pol,G8,a;
G8 := GF(2, 8, alpha^8+alpha^4+alpha^3+alpha^2+1):
a := G8:-ConvertIn(alpha):
exp := 0;
pol := G8:-input(dec);
fin := G8:-`^`(a,0);
if dec = 0 then
  exp := -1;
else
  while pol<>fin do
    pol := G8:-`/`(pol,a);
    exp := exp+1;
  end do:
end if;
return exp;
end proc:

#関数名：gen_AestheticQRcode
#入力1：Vec(生成したQRコードが格納されたベクトル)
#入力2：image_path(元画像のパス)
#出力：AestheticQRcodeが格納された二次元リスト
```

```
123
124 gen_AestheticQRcode := proc(QRcode_vec,image_path);
125 local i, j,image,resize_height,resize_width,resize_image,AestheticQRcode;
126 local resize_gray_image,avarage_luminance,epsilon;
127
128 image := Read(image_path);
129 resize_height := 21/Height(image);
130 resize_width := 21/Width(image);
131 resize_image := Scale(image,resize_height,resize_width);
132 resize_gray_image := RGBtoGray(resize_image);
133
134 avarage_luminance := 0;
135 for i from round(21/4) by 1 to round(21*3/4) do
136   for j from round(21/4) by 1 to round(21*3/4) do
137     avarage_luminance := avarage_luminance + resize_gray_image[i][j];
138   end do;
139 end do;
140 avarage_luminance := avarage_luminance/((round(21*3/4) - round(21/4))^2);
141
142 AestheticQRcode := RGBtoYUV(resize_image);
143 epsilon := 0.25;
144
145 for i from 0 by 1 to 21-1 do
146   for j from 1 by 1 to 21 do
147     if QRcode_vec[i*21+j] = "Black" then
148       if AestheticQRcode[i+1,j,1] >= avarage_luminance - epsilon then
149         AestheticQRcode[i+1,j,1] := avarage_luminance - epsilon;
150       end if;
151     elif QRcode_vec[i*21+j] = "White" then
152       if AestheticQRcode[i+1,j,1] <= avarage_luminance + epsilon then
153         AestheticQRcode[i+1,j,1] := avarage_luminance + epsilon;
154       end if;
155     else
156       #print("エラー！");
157     end if;
158   end do;
159 end do;
160
161 AestheticQRcode := YUVtoRGB(AestheticQRcode);
162 return AestheticQRcode;
163 end proc:
```