芝浦工業大学

博士学位論文

電子透かしによる画像の保護特性に関する研究

平成 26 年 1 月

氏名：　魏　遠玉

目次

第１章　序論

研究背景と目的

近年、インターネットの普及に伴い、またデジタル信号処理技術の進展により、画像、動画などと言ったディジタルコンテンツの複製や改竄や配信などが簡単に出来るようになった。インターネットを利用してデジタル情報をやりとりできるようになり、利用者が大幅に増加し、画像や映像や音楽の配信において、容易に量産したり輸送したりできるという利点があるが、同時にデジタル著作物の再配布を容易にしてしまうという欠点にもなっている。又、画像や動画などの代表的な著作物を再配布可能な環境が既に整っているため、デジタル著作物の再配布を完全に抑止することは容易ではない。

電子透かしとは人の目に見えないマークをコンテンツ(画像、動画など)に情報として埋め込み、必要に応じて検出することによって、コンテンツの著作権保護や真偽判定に役立つ情報セキュリティ技術である。情報を埋め込み、著作権の所在を主張しようと企画したものである。また、利用者の情報を埋め込む事によって、不正行為が行われた際に、埋め込まれている情報を検出する事で、不正行為の防止に役立ち、著作権侵害に対する検証方法として利用されている。現実には、電子透かしの実用化を必要とする製品が多く、製品に電子透かしを埋め込むと容量の増加、画質の劣化、情報耐性の確保という課題を解決する必要がある。

又、電子透かし実用化の問題は、各種指摘されている。その後、技術の開発、応用の変化、枠組みの見直しなどにより、状況は変化していると考えられる。電子透かしは認証という観点では、暗号方式と対比されるが、暗号化データが改変なく送受されるのに対し、画像はあるレベルの改変を行っても同一と見做せるという違いがあり、技術の適用対象が異なることを認識する必要がある。図 1-1 は認証などの証拠物として使用されているものを、存在確率を横軸にして並べたものである。暗号においても計算量的に安全が確保されているだけであり、安全を確定することは証明できないことが指摘できる。また、SSL などの実用化レベルでは、128 ビット程度の安全ではないものが使用されているが、日常生活レベルでは、使用が停止されるようなことなく運用が継続されている。これから、単独では安全性の保証が証明されていない技術でも、それを使用する場面である状況の制限や、解読のコストと解読による不正収益のバランスなどにより、実用化がなされることを読み取ることができる。

電子透かしにおいても、技術的な認証性をメディアだけ切り離し単独に評価する手法を適用するのでなく、使用形態の環境下において、状況を限定し他の技術との融合を図る事により、存在価値が生じることが示唆されている。画像を単独で扱うのでなく、ビューアーと連携して扱うとか、ハードウェアのビューアーとリンクして扱う、ハードウェアキーを使用するという考えもある。融合の例としては、e-forensics の一環として、写真のデータ

解析からカメラの機種を推定する方式や、写真データを改竄したときの変化状況の推定手法が発表されている。応用の変化として、印刷画像への電子透かしの埋込みがある。これは、雑誌などの写真に所定の検出ソフトを作用させると割引情報が取得され、これがクーポン券となるサービスである。このサービス形態は電子透かしに厳密で無限の責任を果たす認証を行うのでなく、責任範囲が有限で制限を設定できるような応用である。メディアに対しコストをかけて解読して不正利益をあげる対象になりにくいという応用である。このように、厳密性を回避した中間的な認証機能を果たす応用には好適な結果をもたらすことが分かる。システム化の例としては企業内イントラネットでのみ流通させるという閉鎖的なやり方もある。

メディアの著作権確保という当初の目的を発揮するため、メディアの登録という手法がなされている。しかしこの手法も登録を原因とした副作用があるため、必ずしも完全ではなく、問題は解決しているわけではない。これについては、登録のプロトコルの詳細化などについて、今後も検討がなされると期待される。

電子透かしを無効にするアタックの中で、最も根本的に無効にすると論じられてきたものに Inversion Attack がある。しかし、理論的に又、数式上で議論することにより、この Inversion Attack が成立するという議論をしてきたが、実際の埋込み処理を演算により行うと近年の複雑な方式においては、量子化誤差が伴うことから、Inversion Attack は難しいことが指摘されている。電子透かし単独の耐性も向上しつつある。電子透かしの埋込み情報量を削減すれば、埋込みビットは冗長性を有し、耐性を向上できる。

電子透かしの検出器を難読化する意義は、難読化により、著作権情報等を検出するときに事実上公開される検出器の動作が隠蔽され、埋込み検出の一連の動作も隠蔽され続けることにある。隠蔽されることによって、検出器の動作を解析することによる攻撃は直接的にはできなくなる。これにより、パラメータの変更でユーザーごとに異なる電子透かし（Fingerprinting）を埋込んだまま、埋込みシステムの基本構成を長期に変更すること無く運用できることになる。この効果は、プライベートな電子透かしにおいても、耐性向上の一つとして有効である。画像流通に際してセキュリティや履歴による立証のため電子透かしを使用することも検討されている。

本研究では、検出器を公開領域に出し、プログラムの難読化によりその領域での認証性を向上させようとすることと、埋め込み手法の高能率化を図ることが目標である。検出器ソフトは難読化を施し、鍵に相当する「情報の場合の数」が膨大であり、探索するには計算量が多くなるということにより、途中結果の数値に基づいて対応させた素数を復号鍵とし、次の処理に必須の定数値を復号鍵に対応する公開鍵で暗号化して、難読化の要素として、計算量を増大し、解析時間もかかる構成となっている。又、電子透かしは、攻撃がある状況では、確定的な認証性は得られていないが、他のデジタル認証手段と比較して確率的には、同等レベルの証拠的価値があると考えら、その存在価値はあるものと考える。

図 1-2 は従来の閉鎖型電子透かしシステムである。これに対する図 1-3 は提案する難読

化を施した公開型電子透かしシステムと一般のソフトウェア全体に適用が可能である例証が得られたことを示すものである。この構成を実現するための難読化の方式検討を行っていく。又、電子透かしの耐性を向上させるため 1D、2D、3D 変換後に埋め込む方式の検討を行う。まず、1D、2D、3D 変換の電子透かしを構成し、極力公平な比較を行っていく。極力という意味は次元が異なると埋込ビット数が変化し、直接的な比較ができなくなるため、埋込ビット数を揃えるとともに、揃わない処は公正な換算により合わせるという手法をとっていくことである。例えば、3D 情報を 3D 直交変換した後に交流成分の第 2 項となる要素は 7 か所あり、１次元あたり 2.33 か所と端数が出るので、調整が必要である。本研究では従来の 3D－DCT 方式を比較対象としながら構成していく。又、他の変換である Chirp 変換をとり込み有効性を調べていく。

以下の章では、まず、第２章で電子透かしの検出器を難読化してその動作を隠蔽する方法について述べる。秘密情報鍵としては、実行時に出現する変数の値を復号鍵とし、対応する暗号鍵を求める。実行時に使用する他の必須数字を暗号化して隠蔽する。この暗号化は、いわゆる公開鍵暗号方式を用いる。これにより復号鍵、暗号鍵の長さを長くすれば、解読は困難であるように設定できる。実際にいくつかの数で暗号化、復号を行い、数値の増大とともに演算時間が増加していくことを検証していく。また、この方式を一般化し、実用化を行った。第３章では、３次元線形変換を行う電子透かし方式を検討する。１次元、２次元変換の電子透かし方式の研究は多いが、３次元変換を行う方式はまだ少ない。本論文では、１次元２次元３次元の性能比較を極力公平な形で行っていく。埋め込み方式に対し、埋込み手法の高能率化を図るために、離散コサイン変換（DCT）に加えて Chirp 変換と DCT 変換を組み合わせる手法を検討していく。電子透かし方式としての総合的な特性の評価を行っていく。第４章では、その他の実験について述べる。第２、３でも実験を行って検証するが、精度検証や S/N 値の換算手法などについて述べる。第５章にまとめと結論、今後の課題について述べる。

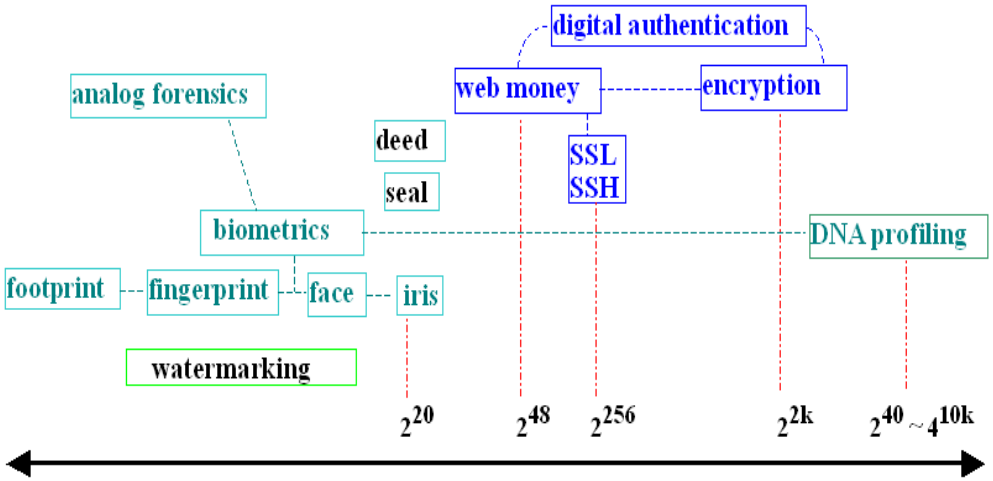


図 1-1　アナログとデジタルの証拠物の例.

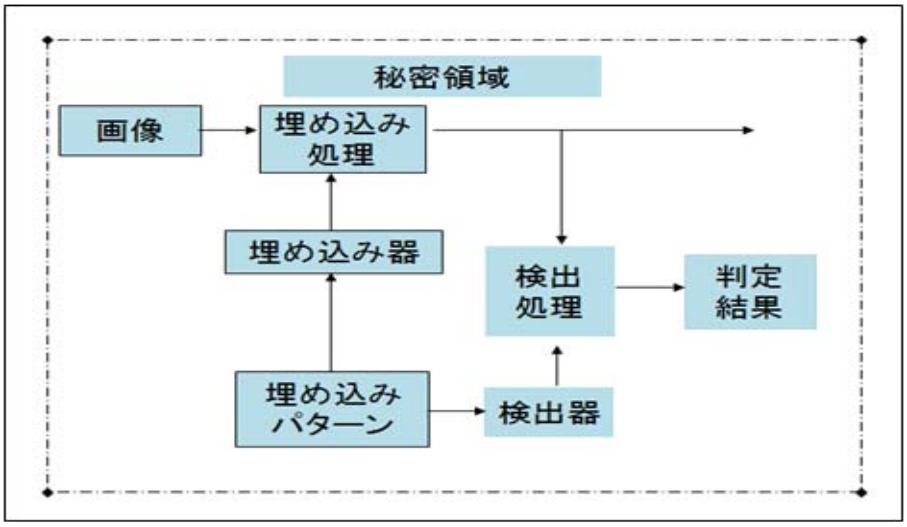


図 1-2　従来閉鎖電子透かしシステム.

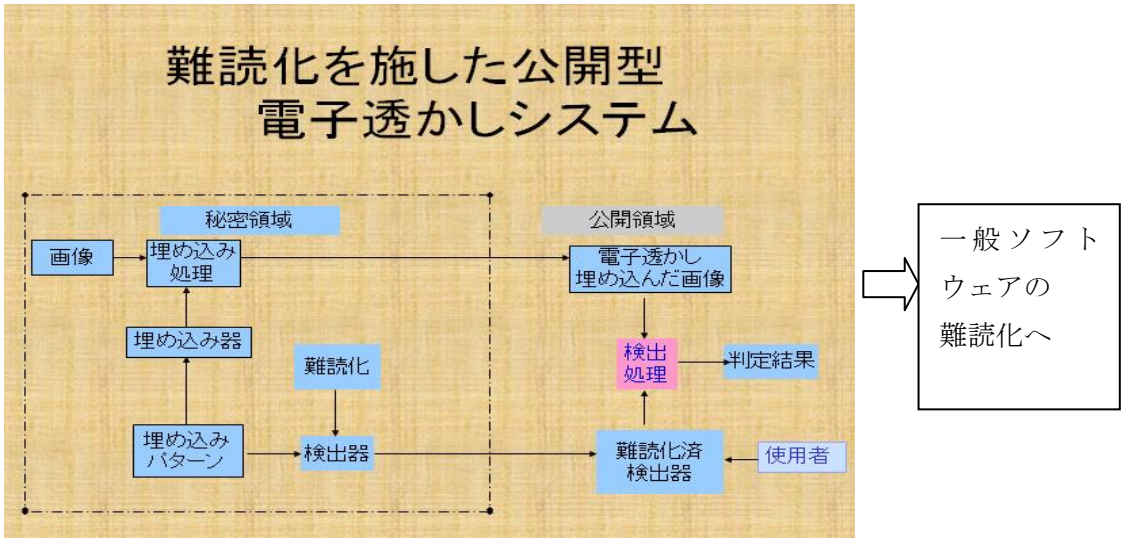


図 1-3　難読化を施した公開型電子透かしシステム.

第２章　電子透かしの難読化

２－１はじめに

電子透かしは画像に対してソフトウェアプログラムで、マークとなる透かし情報を埋め込む処理を行い、埋め込まれた情報の検出処理を行う。ソフトウェアで透かしを埋め込み、検出処理を行う時、そのソフトウェアプログラムを解析することによって埋め込みの方式や検出のアルゴリズムがわかる可能性がある。これを攻撃者による攻撃と呼んでいる。この攻撃により、電子透かしを改変したり、無効にしたり、また完全に取り除くことも、攻撃者が精力的に解析を行えば可能である。どの程度まで難読化を行えば十分であるか、ということは、このように攻撃者の行動にも依存する。しかし、研究レベルでは、そのような人間に行動の要素は予測が難しいこともあり、考慮に入れないで、あらゆる攻撃に対して完全に解読が不可能であることや、公開鍵暗号のように計算量的に評価することが重要であると考えられる。

一方、第 1 章の図 1-1 に示した図でもふれたように、実社会では、特に商業的には、セキュリティの観点からは万全でないものも実用化されている。たとえば、web マネーは 16 桁の英数字とすれば、128 ビット程度の種類となる。2^{128} の種類の探索自体は、今の計算機で可能な範囲に入っているが、実際に探索する場合は、何回かエラーが続けば、試行を停止するなどの処置がなされ、探索が多数回できないようなシステム上の枠組みなどがなされている。すなわち web マネーそのものの安全性は不完全でも、それを取り巻く環境を合わせたシステムとして考えると、十分安全性が保たれるような形になっている。

このような事例を観察すれば、電子透かしの検出器を難読化することは、電子透かしという画像メディアと検出プログラムというソフトウェアという異質のものが組み合わさっており、この組み合わせを一種のシステムと考えると、安全性は向上する可能性があることが考えられる。

難読化の研究では、Barak らの難読化は不可能であるという数理的な論文[2-1]が発表されたあと、難読化の研究分野において、研究が低調になった傾向が見られた。すなわち難読化の研究を行っても、難読化は不可能であることが証明されているので、無意味であるとの考えが強まったと考えられる。しかし、Barak の論文の証明では、限定された連続数上での無限個の点の上に関数を定義しており、現実世界は有限の離散的な対応関係しかない点が、数は減るが異なる点であることが懸念される。無限は有限を含むことは確かだが、Barak の証明には無限の要素を論理に使用しており、これが現実世界と遊離している点であること問題ではないかと考えられている。実際、Barak 自身が全てを対象としてないことを、自己解説で述べている。そこで、この枠組みを外れた個別の難読化を行う事は依然として 意味があると考えられる。一方、難読化や暗号化処理は、途中の過程で機密情報の演算処理

が観察できることから、公開領域での演算においても機密が保たれるよう公開領域（White-Box ）での処理の必要性があり、あえて White-Box ということを条件に課した難読化方式の提案がある[2-2]。難読化処理はその前後でソフトウェアプログラムの動作結果が同じであることや、暗号手法のようにセキュリティ性を証明することが理想である。難読化は英語では obfuscation と呼び、ここではそれを使用している。

電子透かしの検出器ソフトウェアの難読化は、実際に使用する必要性があるのは、提案する検出器公開型の電子透かしや２段階電子透かし方式では、著作権争議になり検出を確認する時であると見做せば、通常は使用されないままとなり、その動作速度が低速であっても不便とならない。本研究では、従来が行ってきた計算量が増加することを許容した形で、計算量のみに依存する難読化を確実に行う手法[2-3]、計算量を正確に評価しながら、計算量的上の仕様の設定可能な難読化手法を詳細化し、実用化を促進するための関数化への一般化、また、乱数データの混在化による方式の検討を行う。提案方式においても課題が残るが、計算量負荷を確実に設定できる点が特徴である。

2-2　難読化の従来技術

2-2-1　従来技術

ソフトウェアの保護、再利用などのためには、解析や理解の容易なプログラムを作成することが重要である。しかし、システムの安全性の確保や知的財産権の保護などのために、プログラムを多数のユーザに配布する際に、ソフトウェアのプログラムのソースコードの解析をされたくない場合がある。そのような場合には、解析が簡易なように作成したプログラムを、解析が困難になるように変換する方式が必要であると考えられる。このようなプログラムの等価変換を、プログラムの難読化と呼ぶ。門田らはループを含むプログラム（図2-1a）を自動的に難読化する2通りの方法を提案し、それぞれの方法の有効性を評価するための実験について報告している[2-4]。実験の結果、極めて小規模なプログラムに対しても、提案する方法が有効であると述べている。大規模なプログラムは、一般に、元より複雑であることもあり、変数や関数の名前を意味のない文字列に置き換えたり、コメント文を削除したりと言う単純な方法を適用するだけでも解析が困難になる[2-4]。

難読化の種類により、分岐、誤差付き変換、計算式の複雑化、暗号化の難読化がある[2-5、6、7]、以下４つの項目に分けて紹介する。

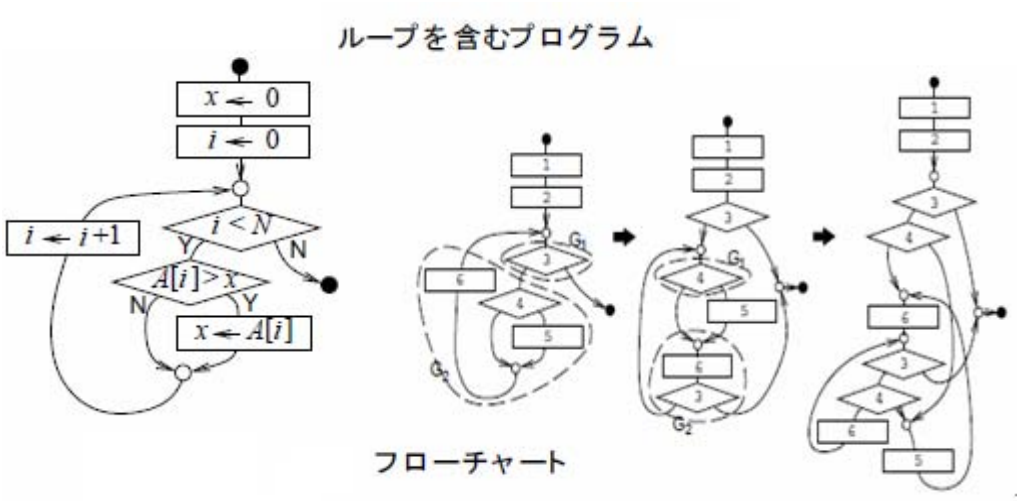


図 2-1a　ループを含むプログラム.

1） 条件分岐の難読化

条件分岐文(主に if 文)を多く使用する事でプログラムを読み難くするというのが基本的な考え方である。しかし、検出ソフトには、一回で正しい結果が得られれば、それで if の分岐先が判明する。また、ソフトを全体的に検出すれば、if 文を加えても、はじめの加算の演算に条件が存在しない以上、意味のある条件分岐は発生しない。これはある演算が確定的な別の演算に対応した形で変換されたことによっているため、処理後の値は難読化する前と後で同一になり、全ての結果が分かるところが問題であると考えられる。また、分岐の数も限界が有り、解読は簡易と考えられる。

2） 誤差付き変換の難読化

実数を区間分割し、実数の四則演算を関数と定義し、入力となる変数値に対し、あらかじめ関数値を計算しておき対応表を作成する。関数演算を対応表に置き換えると、関数アルゴリズムは、変数の区間と関数値の対応を定義できる。このアルゴリズムの区間ごとの対応表により、計算された結果は真の解を含み、その区間内の関数値の変動幅が誤差となる(図 2-1b)。このように変換すれば、１つの演算が、複数種類の演算に増加するため、難読度は確実に向上したとする方式である。検出は量子化された値について処理するため、ある程度幅を持たせることができる。そのため、検出に影響の出ない程度の変換をほどこせることになる。 たとえば、量子化値から DCT 逆変換をする演算をこの関数により、対応表で置き換えると、適正な量子化値に対しては、一定の誤差を含むが正しい結果を出力する。しかし、不要な量子化値に対しては、間違った値を出力する。したがって、攻撃者

が関数を解読しようとして、入力と出力の関係から関数の対応を求めると、試行を増やしても正解に至らない。

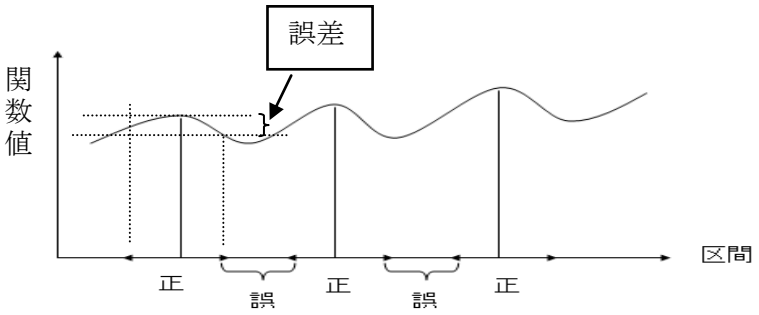


図 2-1b　誤差付き変換関数の例.

3） 複雑な計算への変換

複雑な計算への変換は、例えば、多項式の計算式に対して行える。ある問題を解決するために、複数のアルゴリズムを用いることができるとすれば、どのアルゴリズムを選択すれば最も効率的に処理できるかを判断するための判断基準の一つとして、計算複雑性を利用することができる。これを逆に利用して同じ計算を複雑な方法で行うようにすればよい。なお、実用上のソフトウェアにおいては、アルゴリズムの計算複雑性に加えて、プログラムを実行するハードウェアの環境や、プログラム上でのアルゴリズムの実装方法などが複雑になっている。計算複雑性理論により機械的な処理では、自動的に変換ができないので、人手による変換が必要である。

例えば　　Y＝a＋b　に対し、

$$a+b\rightarrow \begin{cases} x = a^3+b^3+3a^2b+3ab^2 \\ Y=\sqrt[3]{x} \end{cases}$$

とすると複雑な形となっている。

4） 暗号化

暗号化と復号化は短時間に少ない手間で実行でき、逆に解読には時間がかかることや手間が必要になることが理想である。そのために、暗復号化のアルゴリズムを複雑にし、逆に演算ができないようにし、鍵を類推しにくくするなどの工夫が必要である。暗号・復号のアルゴリズムは簡易であるので、プログラムとして実装することも可能である。

公開鍵暗号は暗号化鍵と復号化鍵が異なる暗号のうち、一方から他方を容易に算出することが非常に困難である方式で、鍵の一方を公開することができる。現在の代表的な暗号方式に、RAS 公開鍵暗号がある[2-9]。まず、平文を整数とする。整数 N を決めておく。そして、平文を整数 N 未満になるようにブロック化する。１つの平文ブロックを P とする。暗号処理では、平文ブロック P を暗号文 K に変換する。この変換で、P を e 乗して N で割

った余りをKとする。平文全体に対して繰り返し行う。復号化処理では、暗号文ブロックKを平文ブロックPに変換する。この変換では、Kをd乗かけてNで割った余りをPとする。暗号文全体にたいして繰り返し行う。ここで、e、d、Nが鍵である。

暗号化　　K＝　P^e　(mod N)

復号化　　P=　K^d　(mod N)

鍵e、dは指数として使用する。 dを指数鍵（秘密鍵）と言う。鍵e 、Nを法(モジュロ)とした公開鍵として使用する。鍵　e、d、Nの間には次の関係がある。

N＝p*q

e*d(mod lcm((p-1),(q-1)))=1

pとqは、異なる素数である。Nはpとqの積である。lcmは(p-1)(q-1)の最小公倍数である。eとdの積をNで割り余りが１になるので、eとdはお互いに逆数の関係にある。秘密鍵を求めるためには公開鍵を因数分解して素数p、qを求める必要があるが、それが解法が見つかっていないため、非常に難しい。p、q が十分大きいと、適当な素数といっても困難なことであるから、計算量的に安全である。

暗号プログラムの一般的な問題点は３つある。

①安全性を保証することが難しい

②大きなデータを扱うことが難しい

③速度を十分なものとすることが難しい

暗号化にとって、安全性の保証には前記の素数の長さを大きくとり、十分な時間と費用をかける必要がある。暗号化で処理するデータは年々大きくなりつつあるので、鍵も大きくなって、1024ビットで処理することも危険と言われている。今後、2048ビットで処理しなければならなくなる。するとテストにも時間もかかるし、デバックで変数の値を調べるのも大変なことになる。暗号をストリーム配信などに応用する場合はさらに処理速度が要求されるので、プログラミング技術よりも数学的な計算量の確認とその最適化の技術が必要になる。

暗号化方式の難読化では、プログラムを部分的に暗号化することができる。しかし、暗号化したプログラムは配布の過程での解析が困難であるが、暗号化されたままでは、通常計算機が実行できないので、実行されるまでに必ず復号される。多数のユーザに配布され各計算機で実行される場合、復号後に残ったプログラムから解析される危険性が残る。暗号化されたプログラムより解析が容易である。難読化したプログラムも暗号化したプログラムも無制限に時間を費やせば解析できないとは限らない。なお、プログラムを変換すると、仕様を変えなくても、大きさが増したり、実行効率が低くなったりすることもある。暗号化の大きな弱点は、保護されたメモリにプログラムを復号しない限り、弱点を完全に解決することはできない点である。

2-2-2　難読化の従来技術（2）

Collberg は文献[2-4]で難読化技術の調査を行った結果の解説を行っている。多数の手法が列挙され、それぞれの特徴が書かれている。以下に、その中から、主要と考えられるものをいくつか取り上げておく。

・配置の変更（5.5　Layout Transformations）、
・制御の変換（6　Control Transformations）、
・まぎらわしい記述（6.1　Opaque Predicates）意味なく、If 文を追加する。
・計算法の変更（6.2 Computation Transformations）$a+b \rightarrow a^2+b^2+2ab$
・loop 条件の拡張（6.2.2　Extend Loop Conditions）
・関数を使用しないで、冗長な演算の追加（6.2.6　Add Redundant Operands）

X=y+v →　x＝x＋v＊p （p＝1）

など多数がある。しかし、これらはプログラムを読みにくくするという効果あるが、動作解析コンパイラのオプテイマイゼーションで（最適化）を適用すれば、使用していない変数、無駄な変数を除去できることや変更のパターンを捜索することができるなどのため、理論的に難しさを評価することができない、と言う問題があった。

2-3　提案技術

2-3-１　難読化の構成

以上の従来例の問題点の一部を改善するため、本研究ではまず数量的に評価できる方式を考える方式を述べ、詳細化し、更にランダム関数や ROM での実現性を調べることにした。

難読化では逆コンパイラのようなもので戻らない、理論的に検証された意味的に難読化した上で、計算量を評価するのが理想である。ここでは、工学的観点からまず意味的な部分は保留したまま、計算量だけ評価することと、計算量を増加させることによる難読化を行うことを考える。難読化されたものを解読するのに一定の計算量がかかることを規定することは重要である。そのため、あるプログラムを実行する時に一定以上の手続きが要することを証明できれば、難読化の要素技術の一つとして、利用可能となる。計算途中の結果から復号鍵を決め、次の処理に必要な数値に対して対応する暗号鍵で暗号化を行うことにより、処理の流れを規制する。これにより処理の並列化や処理の省略を行うことができなくなる。従って、上記途中結果を求めて復号鍵を求めた後、復号処理を行って先に進む必要ができ、処理量を確実に設定できる様になる。

電子透かしの検出器ソフトウェアの難読化は、実際に使用する必要性がある。著作権争議になり検出を確認する時に必要になると見做せば、通常は使用されないままとなり、その動作速度が低速であっても不便とならない。本研究では、計算量が増加することを許容した形で、計算量のみに依存する難読化を確実に行う手法を検討する。本研究では、難読手法のうち、まずリバースエンジニアリングにより容易にソースコードに戻るコンパイラの様なものでない意味的な難読化の部分を除外し、埋め込み者が作る検出器を難読化して、公開し、動作はするが、処理のアルゴリズムが解析しにくい形にすることを目指す。計算量を正確に評価しながら、計算量的に仕様の設定可能な難読化手法を構築し、具体的な計算量の評価や見積例を示す。暗号化の手法を組み込み、確実な計算回数の増加を図る。ユニットを何回埋込むかの仕様から、反復して、必須数字の隠蔽回数を決めていく(図 2-3)。

プログラムの実行の途中結果の数値に bais 値を加え素数化して復号鍵とし、以後の計算に必須な数値を暗号化してプログラムの書き換えをする(図 2-4)。通常の実行も遅くなる点と、一回実行すると、復号鍵は全て入手できる点という問題が残るが、計算量負荷を確実に設定できる点がメリットである。また、透かしを埋め込んだ画像と共に検出器を公開して、公開領域で透かしの有無を検出することにより、検出を公開領域まで拡大したという意味の認証性が向上する。

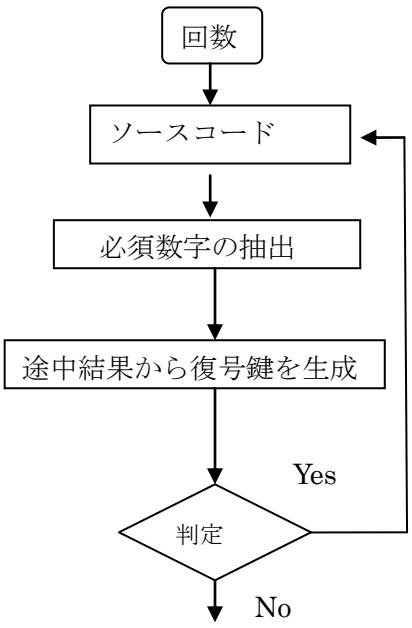


図 2-3 難読化の構成１.

注：回数は何か所でプログラムの難読化を施すかである。判定は復号鍵で次のソースコードへ進むことである。

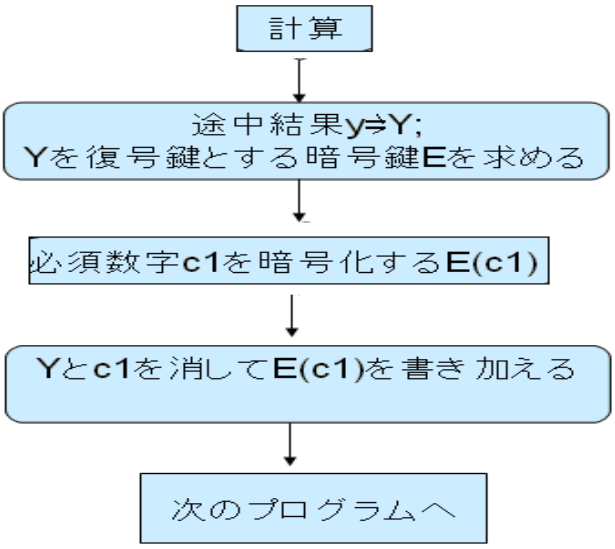


図 2-4 難読化の構成２.

2－4 途中結果を復号鍵とする方式

難読化されたものを解読するのに一定の計算量がかかることを明確化することは重要である。そのため、あるプログラムを実行する時に一定以上の手続きを要することを証明できれば、難読化の要素技術の一つとして、利用可能となる。本研究では、途中結果の数値に基づいて対応させた素数を復号鍵とし、次の処理に必須の定数値を復号鍵に対応する公開鍵で暗号化して置き換え、必須数字の隠蔽を図り、計算量を確実に増大させる方式とする。

図 2-4 にあるように、途中計算結果 y から復号鍵 Y を生成し、Y を復号鍵とする暗号鍵を求める。暗号化をそれ以後の計算に必要な定数値 C1 に施し、E(C1)とする。C1 を削除し、E(C1)を書き加えるという変更を検出プログラムに対して行う。プログラムを実行し、正しい値が得られないと次の計算が進まないため、プログラムの実行をモジュール分解し、並列的に解析することができなくなる。従って、上記途中結果を求めて復号鍵を求めた後のみ、復号処理を行って先に進む必要ができ、処理量を確実に設定できるようになる。

プログラム中の隠した必須数字を別の値に変換し、以降の必須数字に対する演算を変換後のドメインで行えようにプログラムを変換する方法である。データ難読化を計算するプログラムにおける計算途中が暗号化されるにより、途中が切れてしまい、次の計算が進ま

ないことになる。次の計算を出来るように、以下の演算式の変更を行っている。この例では、計算途中 c1 が暗号化され、c1 の値は E(c1)の値に変更する(c1= $E(c1)^d$ mod N)。E(c1)を解いてからのみ、u の計算ができるようになる。(図 2-5)。

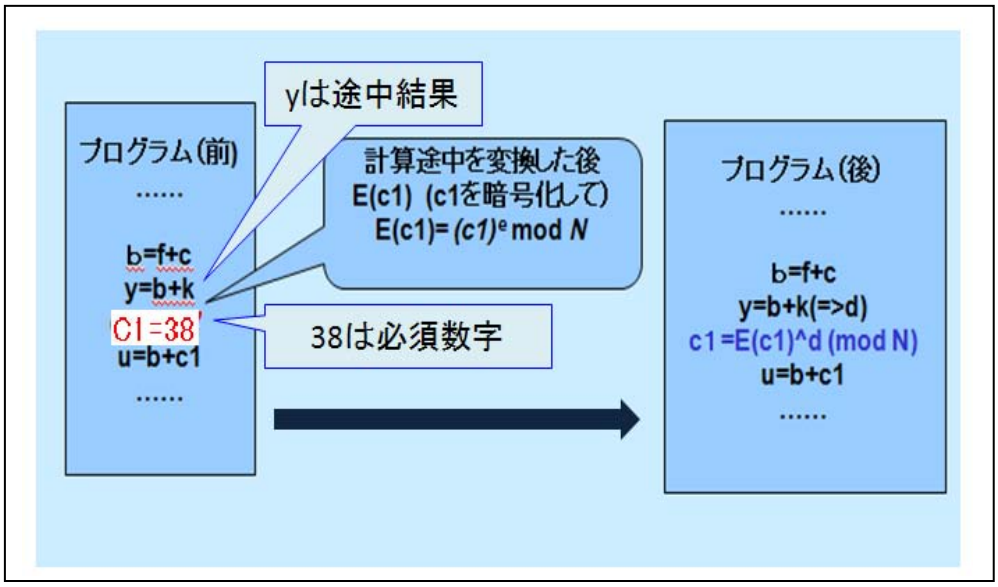


図 2-5　データ難読化を計算するプログラム.

図 2-6 は具体的に途中結果を復号鍵として、必須数字の暗号化を行う方式の変形の構成を示す。ある計算を行い途中結果 y が得られたとする。次に以下の計算に必要な定数、a=38 があり、y と a 等を使って次の B の計算結果を得る。これに対し、まず、途中結果 y=12 に関連する素数 41 を選定する。12 と 41 の差は Bias として、変形後のプログラムに書き加えておく。すなわち、Bias=29 となる。ここで、Bais=29 と以下の例の素数 29 はたまたま一致しているだけで、本来は無関係である。暗号鍵を決めるための素数 29 を基に例えば p=37、q=23 を選び、p-1 と q-1 の最小公倍数 n=396 を求める。また、N=p*q=851 から、暗号鍵 e=29 を得る。メッセージ a=38 を e=29 乗して、暗号結果 $E(a)=38^{29}$ (mod 851)=149 となる。変形後の動作では、途中結果 y=12 に Bias 値 29 を加え復号鍵 d=41 を得る。暗号化されたメッセージ 149 を 41 乗して a=38 を得る。a=38 の消去線はソースコードの一部を隠蔽することを表している。

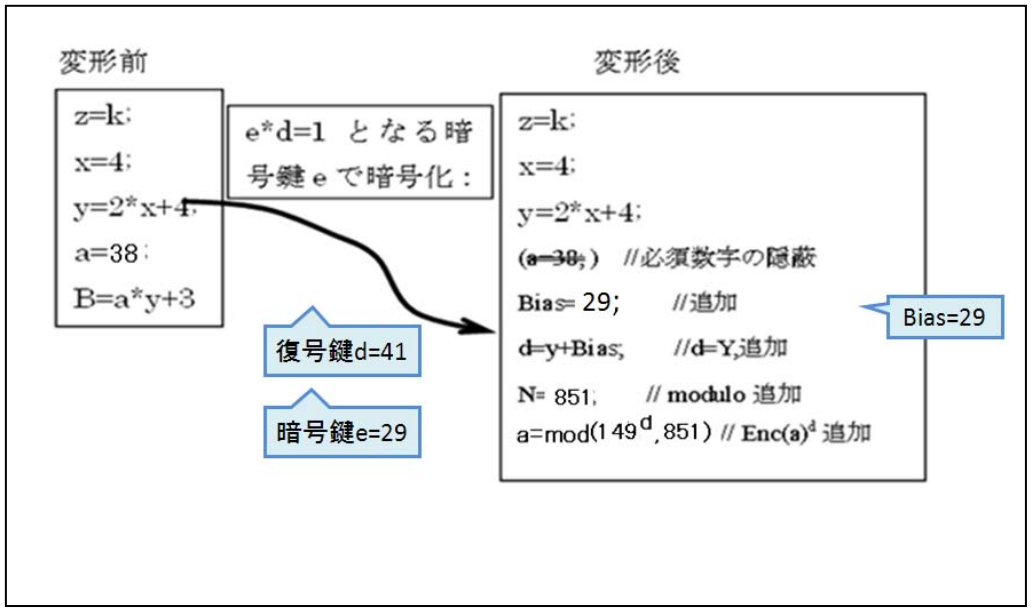


図 2-6 途中結果を使い、必須数字を暗号化する変形.

以上、途中結果から復号鍵を作り、それに対応する暗号鍵を作って、必須数字を暗号化する手法により、検出プログラムを実行するときに必ず行わないとならない手続きを規定できる。具体的には、暗号結果 (a^e)を d 乗する計算する手間がそれにあたる。なお、このような冪乗演算は、モンゴメリー乗算法（モンゴメリ乗算とは、特に時間のかかる除算を実質的に行うことなく、乗算・加減算・シフト演算のみで、高速に整数の積の剰余を求めることのできるアルゴリズムである）で一定の演算量の低減が図れる。

2－5　計算量の評価

難読化の評価は変異を用いるもの[2-9]や、JAVA コードに対する、メソッドとフィールドの呼び出し回数に基づく評価尺度を用いる手法[2-10]がある。また、具体的攻撃法であるスライシングに対応した評価法に基づく設計などが試みられている。これらは、code obfuscation に該当する手法と考えられるが、更に DES 暗号化処理プログラムを対応表に置き換えセキュリティ性を高めた方式もあり、これらは意味的な難読化がなされていると考えられる。計算量は全ての手法に関わる項目である。難読化はその適用前後で、性質が不変であるのが理想で、プログラムの量や動作結果と処理速度が変わらないのが本来の難読化である。しかし、これを必須とすると、拘束条件が厳しくなり、不可能になる場合も生じる[2-1]。そこで難読化前後で性質は有限範囲内まで変更可能という拡張を行えば、攻撃者の対応も多様化するという負荷が増え、実用域も拡張されると考えられる。

計算量の評価には、多項式時間内かどうかを検討することがなされているが、次数に対する相対的な基準になり、絶対量を定めていない。実用的には、ある時点で計算量を絶対

量で規定し、それに対する計算機能力をその時点で対照すれば、解読に要する計算機負荷を見積もることができる。ここでは、電子透かしの検出器の難読化が、高速で常時認証する必要がないものと見なし、むしろ計算量が多い方が望ましいものとして、確実に計算量を増加させることを最初の目標とし、次の節でアルゴリズムを隠蔽する方式についても述べることにする。

計算量のクラスの分類として、多項式時間(Polynomial :P)の範囲内か、それを超越している (Non-Polynomial : NP) かの判定理論がある。理論的に規定できる点が優れているが、無限回数に近いところでの議論であり、計算機で解読していく場合の有限回数から進むことの整合性が悪いので、今回もいわゆる P-NP 問題としての検討は行わなかった。難読化の現実的な計算量評価基準として、「処理結果を得るために必須の、加算・乗算・制御などの処理量」を決め、以下の実験で、その評価量について検討していく。

図 2-6 に示した方式は、途中結果を復号鍵とする暗号化を以下の計算に必須の定数値に施す物である。これは、スライシングの様な並列分解解析が不可能な構成になっている。必須数字が暗号化されているので、復号鍵(d)を求め、暗号化されたデータを d 乗する計算量が主たる計算量となる。この計算量は、d やメッセージ a の語長に依存するため、実際に使用される途中結果は長い語長であることが望ましい。Bias 値は途中結果 y が素数でないときの使用したい素数との差であり、これは、検出プログラム中に公開しておく。この例では、途中結果より大きい最初の素数が採用されたが、単に差だけが示されるだけなので、この値からは、復号鍵 d を推定できないものと考えられる。

表 2-1 に別の事例をあげる。この例では、途中結果がある実数値になった場合で、その有効数字を整数化することを活用し、大きい素数を復号鍵に設定する。小数点以下 5 桁の実数が途中結果の場合、十進で 5 桁のシフトアップを行い、6 桁の整数を得ている。この整数を起点に素数を探索し、素数 319733 を選択したとすると、Bias 値は 149 となる。指数乗数は整数でない物も可能で、例えば 10^5＊1.1 桁と設定すれば、5 桁シフトアップしたものを 1.1 倍し、351542.4 となったものを、四捨五入して 351542 とすることが出来る。また、Bias 値は負の数も可能である。Bias 値は任意に設定可能で、bias 値や大小から復号鍵を推定できない。指数乗数がある場合のプログラム変形例を図 2-7 に示す。

表 2-1　実数の途中結果を復号鍵とする場合.

途中結果	指数乗数	指数乗数後	復号鍵（素数）	Bias	index
3.19584	5 桁	319584	319733	149	i
	5.1 桁	351542	355753	4211	ii
	5.5 桁	479376	479371	-5	iii
	6 桁	3195840	3195869	29	iv

以上の様な方式を想定し、小さい数の復号鍵として、素数を選択した後、対応する RSA 公開鍵を暗号鍵として求めた例を参考のため、表 2-2 に示す。暗号鍵 e と復号鍵 d は、

$$e \bullet d = 1 \pmod{lcm(p-1, q-1)} \qquad (2\text{-}5\text{-}1)$$

なる関係があるので、復号鍵から設計しても、従来の暗号鍵から設計する設計法と同様な手法になる。途中結果と実際の素数値の関係は無くても良いので、復号鍵となる素数は自由に設定できる。設計パラメータである p、q は想定する復号鍵の長さに応じた値を設定すれば良い。また、Bias 値の正負の符号も任意に選択できる。計算量は復号鍵の長さとモジュロ値の大きさによるため、目標とする計算量に合わせ、これらの長さ（桁数）を調整すれば良い。図 2-7 に示すように暗号鍵は公開鍵ではあるが、設計者しか使用しないため、公開する必要はない。そこで、復号鍵を試行により探索することもできない。素数表などから類推する解析を想定した場合は、素数表の最大長付近の素数を d、p、q として選ぶ様にしていけば、N=p*q は当分素因数分解できない大きい数になってくる。

```
z=k;
x=4;
y=2*x+4;
(~~a=3.19584;~~ )  //必須数字の隠蔽
pw=5;              //指数乗数
Bias=149;          //追加
d=y*10^pw+Bias;    //追加
N=modulo(p*q) ;// modulo 追加
a= Enc(a)^d (mod N); // Enc(a)^d 追加
B=a*y+3*z;
```

(a) も変形プログラム

```
z=k;
x=4;
y=2*x+4;
(~~a=3.19584;~~ )  //必須数字の隠蔽
pw=5.1;            //指数乗数
Bias=4211;         //追加
d=y*10^5*1.1+Bias; //追加
N=modulo(p*q) ;// modulo 追加
a= Enc(a)^d (mod N); // Enc(a)^d 追加
B=a*y+3*z;
```

(b)も変形プログラム

図 2-7 実数の途中結果を復号鍵とする場合.

表 2-2 復号鍵と暗号鍵の設計例.

d	e	p	q	n	lcm
13	61	23	37	851	396
17	233	23	37	851	396
19	667	23	37	851	396
29	437	23	37	851	396
31	511	23	37	851	396
41	425	23	37	851	396
43	571	23	37	851	396
47	455	23	37	851	396

計算量的に負荷を目標仕様に応じて増加させた難読化ソフトウェアの変換の流れを図2-8に示す。上記で説明した難読化変形処理は、途中結果が得られる度に1ユニット生成できる。途中結果の出現回数の範囲内で、設定したい計算量に合わせこのユニットを所定回数分繰り返す。初めのプログラムに陽に表示されていた必須数字は暗号化して隠蔽する。

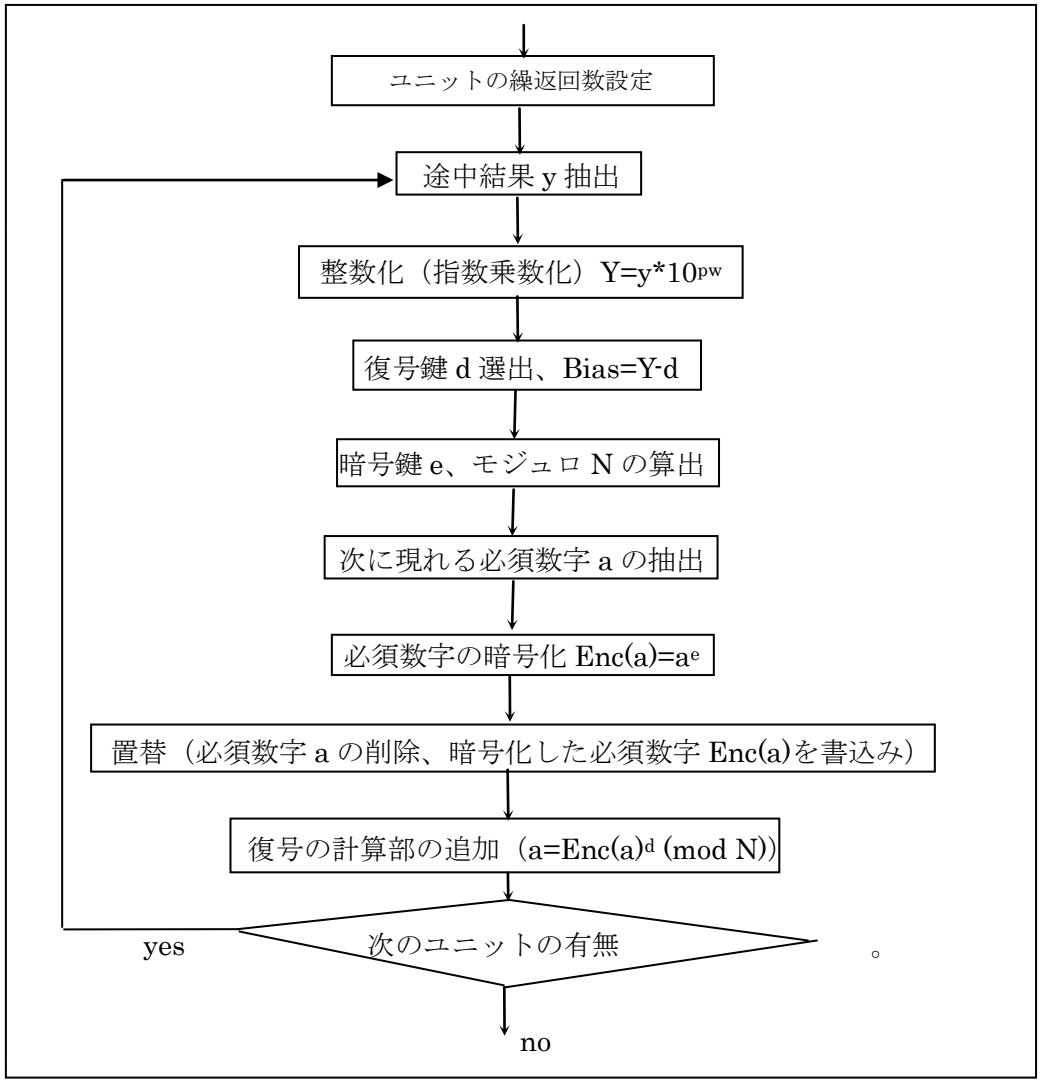


図 2-8　難読化ソフトウェアの変換の流れ.

2-6 難読化を施した検出器公開型電子透かしのシステム

画像に透かしを埋込み公開する一方で、透かしの検出器を公開する。これにより、任意の第３者も透かしを検出する事が出来、公開領域で第三者とともに、透かしの正当性を検証することができる。ただし、当初検出器は同時に公開する方向で検討していたが、通常は検出の必要がないので非公開のままとし、争議などの問題があったときに公開を行うような方式の方がより安全性が高まるので、ここではその点を追加しておく。図 2-9 に検出器公開型の電子透かしシステムの構成図を示す。

画像の電子透かしには種々の攻撃があり、認証性を主張するのが難しい。この方式では、検出器を公開することにより、著作権所有者が内密に検証を行うことに比べて、公開した分だけ、第三者が確認できるようにすれば、認証性の向上があると考える。電子透かしの検出器はソフトウェアプログラムであるが、実行モジュールを難読化してある。実行をして動作解析する攻撃に対応して、計算量的に負荷を増加し、実行速度を遅くして、動作検証をしにくくする。また、意味的な難読化としては、計算方式の複雑化と ROM 関数による表に置き換える方式が検討されている。これは、Chow らの DES 計算の行列によるアフィン変換で表現できるものをテーブル化するものに比べ、非線形の関数も含まれることと、用途において、使用する入力値と使用しない入力値があることを利用して、使用しない入力値の関数出力を偽の値にしたり、離散的な計算を行っているところでは、結果を量子化することになるため、微小な誤差が含まれても最終的には正しい値になることなどの工夫を盛り込むことができる物である。この意味的な難読化を行う方式についての計算量や意味が難しくなっていることの証明はあとの節で述べる、この数節では、計算量を増加する点に絞った形を示す。検出器は計算量的難読化を施した上で、計算量が十分多い形式に変換する方式を採用し、その性能は十分高いと思う。

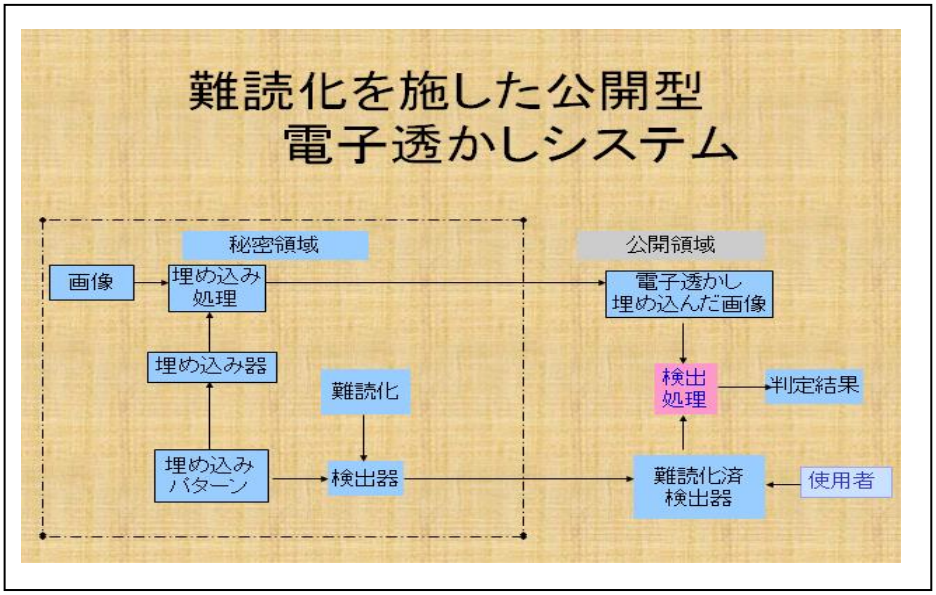


図 2-9　難読化を施した公開型電子透かしシステム.

2-7 実験

計算量を規定する難読化における計算回数の算出と、実行時間例から見積もりを行い、目標とする動作時間に合わせた復号鍵の長さの設計データを作成していく。図 2-10 は、設計する復号鍵の長さに対する暗号鍵を探索した場合の設計時間である。

素数の長さとしては短いものであり、素数表で検索すれば、計算時間は不要だが、計算で求めるような長さの場合の見積もりデータとするため、計算により素数探索を行っている。なお、暗号鍵や p、q の値は、復号鍵に準じた類似の長さのものを用いることとしたが、小さい数から探索する設計手順にしたため、必ずしも統一されていない。

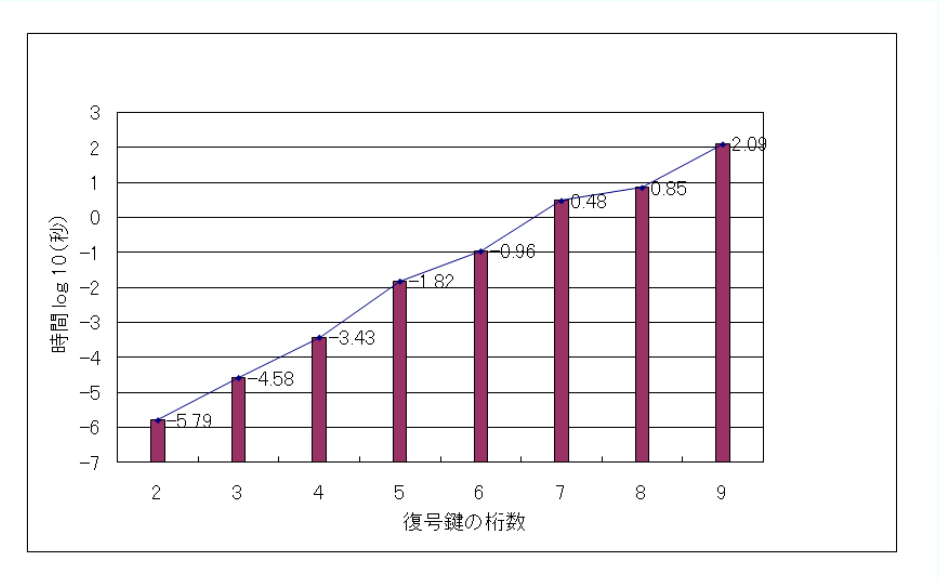


図 2-10　復号鍵の桁数と設計に要する時間の関係.

図 2-11 に復号鍵の長さに対する復号時間を示す。復号時間は、他にメッセージ長も影響するが、メッセージ長は復号鍵と同じ長さを用いた。メッセージは必須数字であるため、その長さは、難読化前の原プログラム次第である。そこで、途中結果 y に対し、指数乗と Bias 値により、希望する大きさの素数に調整することができたが、それと同様な補正を加えることが可能である。即ち、必須数字を指数乗で、整数化し、更に Bias 値で小さい補正を行えばメッセージ長を希望の長さに調整できる。これら調整用の指数乗の数と補正項を変更後の難読化プログラムに記述し、復号後に逆補正を行えば良い。やや段差があるのは、復号鍵、暗号鍵の探索で、見つかったものを使用し、桁の中間位置又は最大位置などを正確に求めてはいないためと考えられる。

復号時間に関しては、十進８桁で約 0.5 秒程度である。使用した計算機仕様は、Pentium4、2.8GHz で C 言語での乗算演算を行った場合で、モンゴメリー乗算等の高速化処理は行って

いない。縦軸は時間の対数であり、ほぼ直線と見做せば、更に長い桁数の場合の結果をグラフを延長して推定することで対応している。

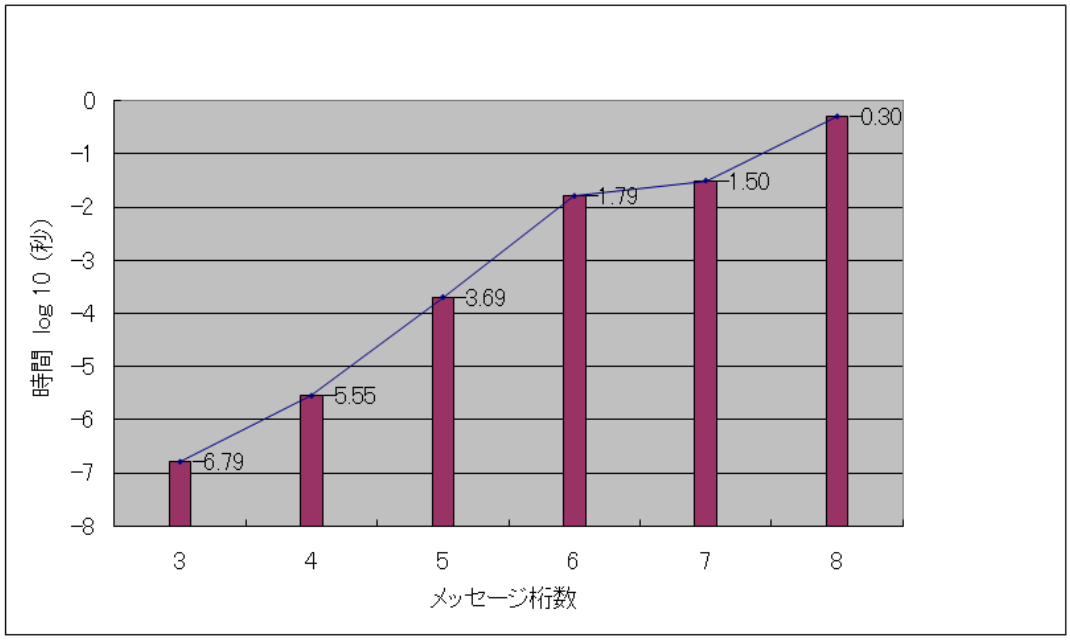


図 2-11 復号鍵の桁数と復号に要する時間の関係.

表 2-3 計算時間と桁数.

計算と桁数

計算と桁数

桁数	1	2	3	4	5	6	7	8
p	7	97	997	9973	99991	999983	9999991	99999989
q	5	89	991	9967	99989	999979	9999973	99999989
N	35	8633	988027	99400891	9998000099	99999640000243	999999600000320	
n	24	8448	986040	16563492	9997800120	999960000396	99999620000280	999899880012000
d	5	79	907	9931	99961	999931	9999937	25956377
e	29	10159	97843	84560371	8325115681	552335018575	86684306711833	38260827373423 13
a	4	78	900	600	6080	600800	6008000	60080000
E(a)	9	6157	306326	12345678	123456789 0	1234567890 12	12345678901234	123456789012345 6

2-8 予測時間

図 2-12 に d、c1 のサイズ（桁数）についての復号演算時間の実測値を示す。横軸は十進数の桁数で、モジュロ N とメッセージ（c1）と復号鍵 d の桁数を示す。そして、c1 は N と近い数である。縦軸は、時間の対数（$\log_{10}$）で、単位は秒で、各メッセージ c1 を d 乗(mod N)する時間を示す。注：破線は時間変化の傾向を延長したものである。

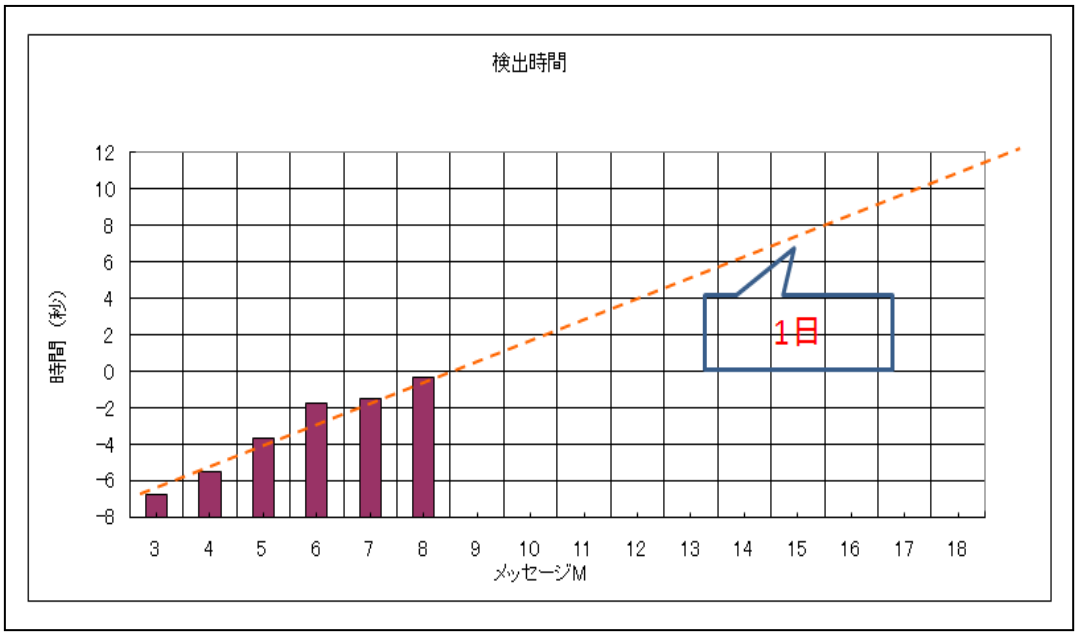


図 2-12　予測時間.

2-9 特殊関数による難読化

難読化の対応となる関数として、表 2-4,2-5 に示すようなものや、一方向性のある関数はより効果が高いと考えられる。既存の特殊関数などで、使用可能なものはないか似ついて調査した。関数の定義式が難解であれば、関数を形状が振動などをしていないものの方が設計者にとっては、使用しやすい。例えば、ガンマ関数などは、解析的に扱うのが難しいため、理論的な解析に当てはめるのが難しいと推測される。このなかでは、ガンマ関数をとりあげ、いくつかのパラメータを与えた後、グラフを書いて比較的低周波数の形になることを確認してみた。一方向性関数とは、多数の変数値の関数値が 1 個の値になり、その 1 個関数値から、もとの変数値がわからなくなることをいう。ここでは、直接的には一方向関数では、ないが、ガンマ関数等を用いれば、関数値から、もとの変数を求める逆ガンマ関数はパラメータが邪魔になって解析しにくいはずであるとの考えで、検討を行った。い

くつかの関数をグラフ化してみて、試行を考えたが、数学的に厳密に考えると、正しい逆関数は求めることができないとしても、近似の関数なら特殊関数でなくても、区分的な多項式を集めても代用が可能であることに気がついた。また、連続関数として考えると、その逆関数を求める問題は困難さがあるが、離散的にすると有限次元の行列、つまり線形変換に戻ってしまうため、離散的にして特殊関数の性質を継承させるのが困難だった。そこで、離散化する前提での検討には無理があるのではないかという結論に至った。

表 2-4 特殊関数の種類.

ガンマ関数	ベータ関数	ゼータ関数
ベッセル関数	球ベッセル関数	エアリー関数
ケルビン関数	シュトルーベ関数	誤差関数
ルジャンドル関数	ベルヌーイ数	スターリング数
フィボナッチ数	積分関数	ポッホハンマー関数
フレネル積分	楕円積分	楕円関数
直交多項式	超幾何関数	二重階乗

表 2-5 楕円積分と楕円関数.

第 1 種完全楕円積分 K(k)
第 2 種完全楕円積分 E(k)
完全楕円積分 K(k)、E(k)
第 3 種完全楕円積分 Π(n、k)
第 1 種不完全楕円積分 F(x、k)
第 1 種不完全楕円積分 F(ϕ、k)
第 2 種不完全楕円積分 E(x、k)
第 2 種不完全楕円積分 E(ϕ、k)
第 3 種不完全楕円積分 Π(x、n、k)
第 3 種不完全楕円積分 Π(ϕ、n、k)

次に、図 2-13 により複雑な関数で非線形変換を行い、それを高次元のマトリックスに配置した変換規則を作り、更にランダムな表にしていく方式を一般化したものを示す[2-6][2-7[2-8]。これは、上記の検討の結果から、離散的な変換では行列ないしは対応表で関数を実現することが良いことがわかったので、その方向で、難易度を高める構想の一つである。各ブロックは、難読化に使用する演算要素で、矢印は進化の方講を示し、より複雑なものに至るようになっている。離散コサイン変換（DCT）やフーリエ変換などの線形変換に対し、非線形変換は行列では表現できないものであり、例えば 2 乗演算子を要素に持つ行列形式の演算子群からなるものなどを想定している。高度マトリックスというのは、そのような演算子を一般化したものを全部集めたものを想定している。アフィン変換は線形変換の一般形で、行列積に定数項を加算したものである。これは、1 次元増加させると、定数項がない、行列積に置き換えられ、同字変換と呼ばれる。関数は、その入力である変数域が定義されているが、電子透かしの応用では、使用しない変域というものが有り、その使用していない部分に間違ったデータを埋込む関数の難読化がある。マトリックステーブルの関数でも、この使用しない入力部分に関数値とは異なる値を入れておくことで、難読化できる。これを全部合わせたものが、図 2-13 のランダムテーブルである。

変換の入力と出力の関係は図 2-14 の下のグラフの様になっている。このような乱数を含んだ解析が不可能なパターンで変換を行えば、解読不能な難読化が実現できる。

非線形の関数を難読化の関数として試行錯誤してきたが、実現に対して離散化するため有限次数の近似行列になるので、その中で関数形を隠蔽した ROM 型を開発し、規模の見積もり等を行った [2—6、7、8]。これについて、次の 2-10 で述べる。

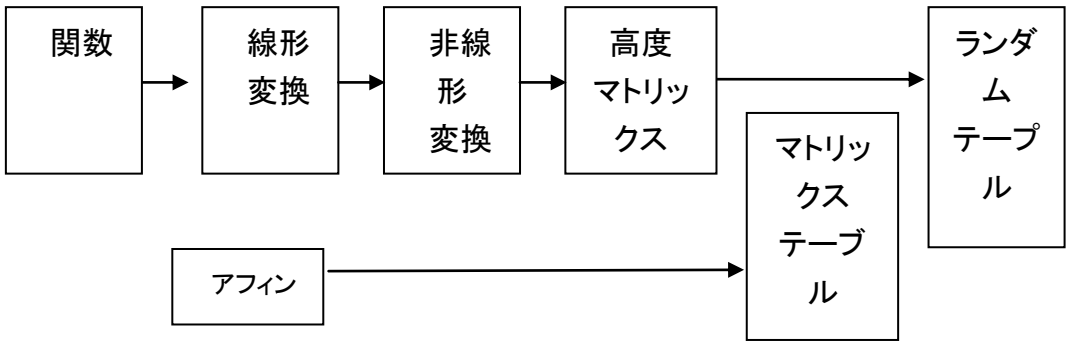


図 2–13　非線形変換による関数の複雑化と離散化の構想.

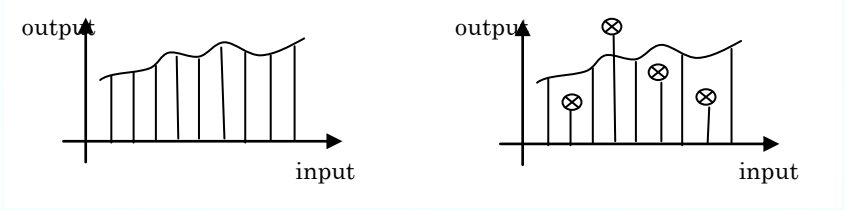


図 2-14 左は従来の表関数関係.　　⊗ 乱数を含んだ表関数.

2-10 ROM による難読化([2-12])

プログラムの難読化は全てのプログラムに適用できる汎用型と、特別な機能を有すプログラムに固有の難読化に分けられる。汎用型にはあらゆる関数類が含まれるが、その中には、ごく単純な関数、例えば

$$f(x) = 3x - 2 \qquad (2\text{-}10\text{-}1)$$

も含まれる。この関数プログラムを難読化した結果に何個かの入力を入れ、出力を下の表１のように得れば、これがｘの一次関数で、係数が３、定数部が-2 であることは、容易に判明する。従って、もとの関数あるいは演算アルゴリズムがあるレベル以下の簡易なものであるとき、どのような難読化を行っても、入出力の関係の観察により、アルゴリズムを推定することは容易にできる。従って、このように容易な計算に難読化を施すのは効果がなく、また、可能性または不可能性を調べても実用性が無い。

一方、従来の電子透かしの検出器を公開する際に難読化する応用があるが、この検出器は変形フーリエ変換を含んでいるため、未知数に関する多次元方程式を解く必要があると考えられる。また、この変形フーリエ変換は、非線形に拡張でき、5 次方程式を組み込めば、一般解法が存在しないものが組み込み可能となる。本論文では、後半で、その構成を述べる予定である。

表 2-6　難読化後の解析試行例.

試行	a	b	c	d	e
入力	0	1	2	3	4
出力	-2	1	4	7	10

Barak の難読化不能な関数の例には、上記(1)式のようなものがあるが、これは、上で述べたように、元来難読化前からアルゴリズムが無い単純な関係でしかない。それを母数を無限の所に一般化と称して持って行って、確率的に解析を不能にしただけで、証明として正しくても、難読化という意味的な例としてふさわしくない。つまり、連続領域では、母数が無限で、いくら試行して、$x=\alpha$ というものにたどり着く確率は、限りなく 0 ではあるが、実用領域での離散変数領域では出力が β になるまで探せば良いので、探索問題としては、有限になり簡易なものである。一次関数の係数を探す問題を基準とすれば、それ以下の 0 次の定数探索といえる。もし、全てのプログラムが、難読化不可能なものと、難読化可能なものに分かれているなら、不可能なものを取り上げて、不可能であるという例示は可能だが、可能なものについての検討はなされていないと考えられる。

もう一点異なる所は、難読化後のプログラムに対し、依然として入力から計算して出力を得るという表現をとっているが、ここで提案する ROM 関数では、計算ではなく表のデー

タを検索するだけで、アルゴリズムが観測される元になるような計算は示されていない。また、Barak の例では、連続関数についての証明であり、従って入力の変数も連続で、無限個ある。しかし、実際の例では、離散的な関数になり、入力の種類は、有限個の例も多い。

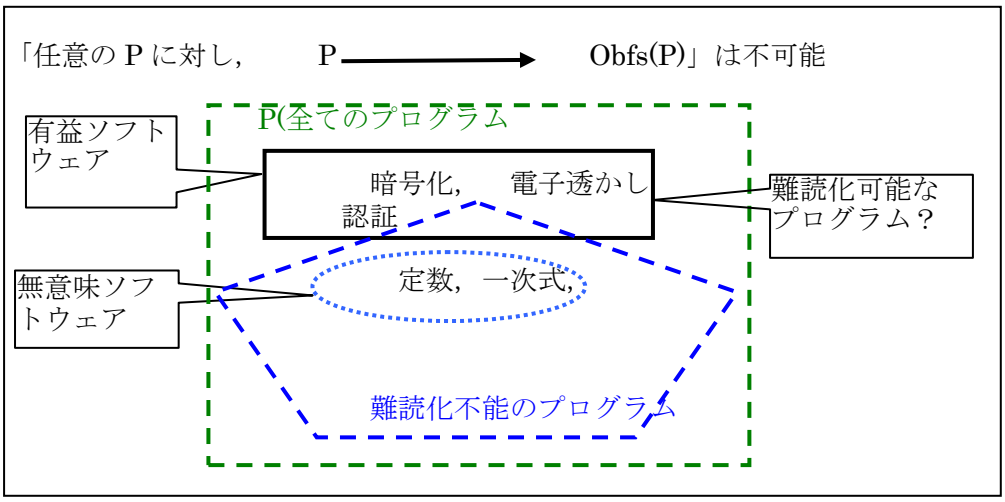


図 2-15　難読化対象の分類.

図 2-16 にある電子透かしの埋込み処理[2-12]を簡略化したものを示す。計算を表にするため、変形離散フーリエ変換の次数を 4 に制限してある。変形離散フーリエ変換（MDFT）は変換係数を変化させることで埋込み手法を隠蔽するものである。入力画像のモノクロ成分 4 画素を 1 ブロックとし、フーリエ変換のような周波数的な領域へ変換する MDFT で変換する。中間周波数に相当する 2 個目又は 3 個目の成分を所定の特性で量子化器 Quant で量子化し、逆変換部 IMDFT で逆変換され、画像に戻す。

図 2-17 に 4 画素のデータの変換を 16 ビットから 24 ビット入力で 2 バイト（16 ビット）出力の 3 個の表により構成する例を示す。これを 4 個の出力に対して 4 種用意する。また、各表の 16 ビットの出力は出力前に値の並べ替えがなされ、この途中結果の値を観測しても関数関係は推定できない。量子化後の逆変換も同様の構成で実行できる。表関数の容量の見積もりを表 2-7 に示す。ROM1 は 128KB で、ROM2、3 が 16MB になる。

量子化器 Quant は最終段の ROM3 に含めることができる。また、ROM1 は ROM2 と統合できるので、変換と量子化までで、1 出力につき、32MB となる。逆変換は、各出力を 8 ビットとして、図 2-17 と同じ構成となり、やはり、ROM1 と ROM2 を統合して、1 出力に対して、32MB となる。以上より、埋込み器の総容量は(32MB+32MB)× 4 出力=256MB となる。検出器の方は、逆変換が不要となるので、32MB× 4 ＝128MB で構成できる。

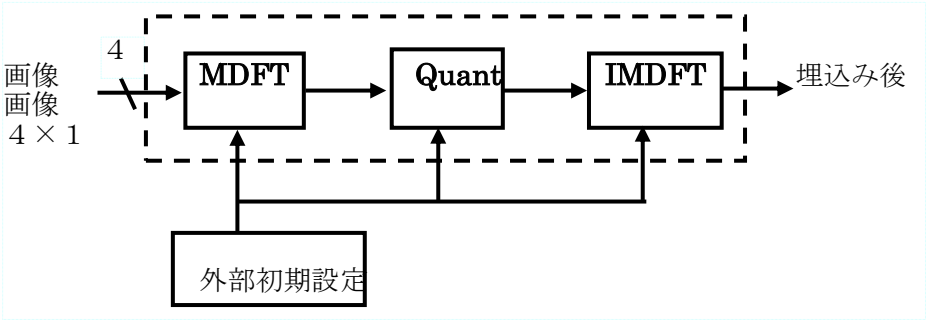


図 2-16　４次電子透かしの埋込み処理.

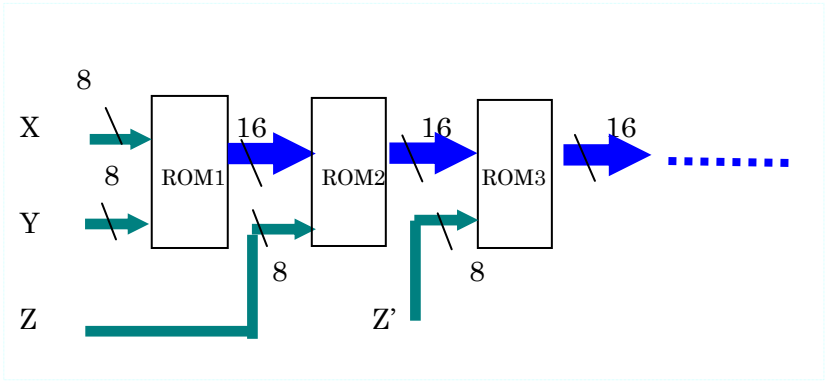


図 2-17　４次変換のデータフロー（検出器の場合、埋込みの場合の前半部に相当）.

表 2-7　データ容量の見積もり.

	入力	ROM1	ROM2	ROM3
入力 アドレス	８×４ （３２）	16	24	24
出力	1	16	16	16
容量 (bits)	2^{16} ＝6553(	2^{16}×16 ＝104857(	2^{24}×16 ＝1677721(	2^{24}×16 ＝1677721(
概数	64KB	128KB	16MB	16MB

上のような電子透かしを埋込むプログラムを作成し、難読化を行う。4 次の MDFT として、(2-10-1)式のようなアダマール変換を変形した実数の変換を行う例を構成する。(2-10-2)式に変形後の変換を、(2-10-3)式にその逆変換を示す。逆変換は小数点以下 3 桁まで表示してある。

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{2-10-1}$$

$$\mathrm{MDFT}_4 = \begin{bmatrix} 1.07 & 0.92 & 1.05 & 0.95 \\ 0.92 & 1.03 & -1.1 & -1.1 \\ 1.05 & -1.1 & -1 & 1.05 \\ 0.95 & -1.1 & 1.05 & -0.9 \end{bmatrix} \tag{2-10-2}$$

$$\mathrm{IMDFT}_4 = \frac{1}{4}\begin{bmatrix} 1.069 & 1.057 & 0.945 & 0.939 \\ 1.057 & 0.871 & -0.918 & -1.019 \\ 0.945 & -0.918 & -1.01 & 0.941 \\ 0.939 & -1.019 & 0.941 & -1.11 \end{bmatrix} \tag{2-10-3}$$

例として、図 2-18 の画像の(199,99)から横に並ぶ 4 画素について、式(2-10-2)の変換を用いる場合を示す。

図 2-18　実験画像例（720×480 画素，モノクロ）.

表 2-8 に埋込み処理結果と検出処理の一部を示す。埋込みは 4 カ所しか無いため、交流成分の平均絶対値最大の第二位置に行う。他に DC に入れる等も可能である。変換後の第二位置に量子化を施すもので、0 は IQ=8 にして埋込みを示している。埋込み後のデータに対し、JPEG 圧縮を行って劣化した結果を c、d に示す。検出は、再度変換のみ行い、第二成分の値が、８の４倍を中心として±16 の範囲にあれば、検出されたと判定する。この場合は、e、f の第二成分が 32 であり、検出ができたことになる。

表 2-8　埋込みと検出データ(a-d は画素値，e-f は変換後の値を示す).

	配列位置	1	2	3	4
a	原画像	244	244	244	244
b	埋込み後	252	252	236	236
c	JPEG 1/5 圧縮	252	252	236	236
d	JPEG 1/10 圧縮	246	247	241	239
e	検出 b	976	32	0	0
f	検出 c	976	32	0	0
g	検出 d	976	13	-3	1

表関数を用いて、計算の手法を隠蔽した電子透かしの埋込み器と検出器を構成した。試行として４次の変換により、例を構成したが、埋込み位置を画像の内側に設定できるため、画像の周辺部を切り取る様な攻撃にも耐性がある。難読化に関しては、表関数によりアルゴリズムの全てを隠蔽した形にすることできた。このあとは、ここに残された手がかりがどれだけあるか、またその手がかりから逆算して、アルゴリズムを解読するのはどのくらい困難かを詳細に調べることは課題として残っている。しかし、この手法を拡張すれば、計算アルゴリズム研究室を表現することなくいわゆる White-Box でのプログラムを動作させる難読化が可能になったことが示された。ROM 容量の限界があるため、直接的に規模を拡大できない問題はあるが、図 2-17 に示したような階層的な構成で、難読性と実用性の妥協点を探ることも可能である。

2-11　難読化の応用

現在、ソースコードの盗用、ソフトウェアの解析や改ざんといった行為が問題となっている。ソフトウェアを保護するためにソフトウェアプロテクションの必要性が高まっている。難読化ツールによってソフトウェアを保護することを目的とした暗号技術を用いた難読化ツールの作成の検討を行った。そこで得られる難読化効果は非常に大きいと考えられる。ユニバーサルロボット株式会社で難読化技術を応用して製品化した。例として以下に示す関数の計算一部のフローチャート示す。

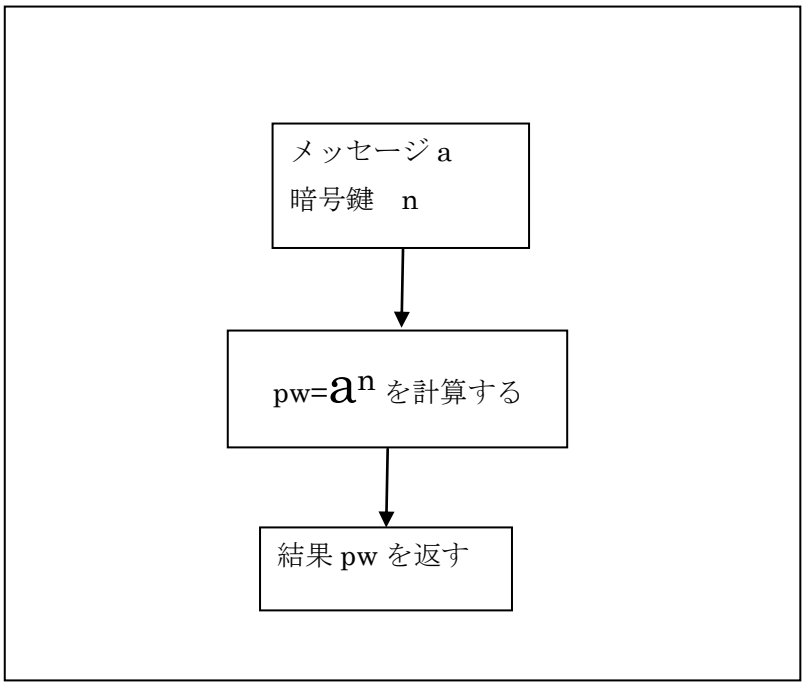


図 2-19　関数化された難読化のフローチャート.

2-12　難読化まとめ

途中結果を復号鍵とする暗号化の方式は、暗号化理論により、確実に計算量が増大する難読化方式である。途中結果を復号鍵とする暗号鍵を求めることができる。十進数 8 桁の計算時間の実測は約 0.5 秒で、計算時間が見積もれるようになった。計算量を見積もる難読化方式に加え、計算アルゴリズムを隠蔽する ROM 方式の難読化方式を検討し、実際に実現構成を行い、モノクロ画像での電子透かし埋込み実験を行った。埋込み領域が小さいため、周囲の切取りにも耐性があることが特徴である。一方、埋込みを拡大するには、ROM 容量の限界が有り、難読性の強さを減らして、階層的な形式にしていくことが考えられる。

第３章 DCT/Chirp ハイブリッド 3D 変換を用いた電子透かし方式

3-1 まえがき

本研究では、従来から線形変換と量子化による電子透かし方式の効率向上の検討を行ってきた。画像符号化では、画像の１次元（1D:one Dimensional ）配列に対する圧縮よりも２次元（2D:two Dimensional ）、３次元（3D:three Dimensional ）の方が高い相関が得られるため、圧縮効率が高くなる。そこで、電子透かしにおいても、より高い次元で高い相関が得てからに埋込みを行えば、埋込み誤差が集中し、元の画像に逆変換した場合に誤差が拡散される効果が期待される。これまでの研究では、１次元より２次元の方が効率が向上していた。そこで、３次元一括で埋込みを行うと、効率が向上すると期待される。

３次元変換を行う電子透かしの方式の従来例として図 3-1 のように、各種あるが、３次元の空間内に埋込む手法は Louizis らにより発表されている[3-1]。しかし埋め込み位置は不特定で次元の比較は明確ではなかった。一方、２D の DCT(Discrete Cosine Transform)後に DC(direct current)係数のみを集めて再度 1D-DCT を行う擬似的な(Pseudo)３D-DCT が Wang らにより発表され[3-2]､以後いくつかの疑似 3D-DCT を用いた関連研究が発表されてきた。8×8×8 画素のブロックに対し３D-DCT を行い､パターンを視覚の性質で分類する方式が Park らにより発表されている[3-3]。3D-DCT と QIM(Quantization Index Modulation)を用いる基本方式が Campisi らにより発表されている[3-4]。又、変換に DWT(Discrete Wavelet Transform)を用いる方式[3-5]、量子化に実数値でのディザ変調（RDM）(Random Dither Modulation)を用いる方式[3-6]などがある。

本論文では、これらの流れの中で、Campisi 等の提案した 3D-DCT と QIM を用いる手法の詳細化を行う。１、２次元との比較と、変換として Chirp 変換を用いる場合についても比較を行っていく。また、変換面での埋込みにより、逆変換時に画像データ範囲外を越える対策として、係数の精度と誤差の大小、検出率の変化を実験し、係数精度を高くする効果を調べた。

3-2 変換の構成

静止画像や動画像の電子透かしについて 3D 変換について各種の研究がなされている。2 次元の離散コサイン変換（DCT）は JPEG(Joint Photographic Experts Group)、MPEG(Moving Picture Experts Group)で使われているが、従来の電子透かしの方式は 2D 変換が主に提案されている。画像の大きさが N×N の２次元の DCT 処理後の結果を F(u、v)とすると、２次元画像信号 f (x、y)に対し DCT は次式によって表される。

$$F(u,v) = \frac{2}{N}c(u)c(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left\{\frac{(2x+1)u}{2N}\pi\right\}\cos\left\{\frac{(2y+1)v}{2N}\pi\right\},$$

$$u=0 \quad \text{のとき} \quad c(u)=\frac{1}{\sqrt{2}}, \quad v=0 \quad \text{のとき} \quad c(v)=\frac{1}{\sqrt{2}} \qquad (3\text{-}2\text{-}1)$$

$$u\neq 0 \quad \text{のとき} \quad c(u)=1, \quad v\neq 0 \quad \text{のとき} \quad c(v)=1$$

また、IDCT(Inverse DCT)の式は

$$f(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} c(u)c(v)F(u,v)\cos\left\{\frac{(2x+1)u}{2N}\pi\right\}\cos\left\{\frac{(2y+1)v}{2N}\pi\right\} \qquad (3\text{-}2\text{-}2)$$

である。２次元 DCT 処理後の DCT 係数の場合、第一個目 F(0、0）が直流成分となり、その他の F(u、v)が交流成分となる。直流成分に近いほど低周波数成分になり、F(N-1、N-1)に近いほど高周波数成分になる。

図3-１は3D変換を用いる電子透かし方式の従来例である。点線から上は、定義通りの後に示す3次元の離散コサイン変換(3D-DCT)を行い、量子化等で、埋込みを行う方式で、点線から下は、定義と異なる3次元変換を配置してある。異なる変換として、2次元変換後に直流成分1個のみを集め、再度1次元変換するものなどがある。Wangらは、ネスト化した、２次元DCTを提案している[3-2]。これは、２次元DCTを行ったあとのDC成分を更にDCTする２重のネスト構造と称する変換を行っている。直流に埋込むと、輝度の変動を引き起し、劣化が大きくなる問題があるため、通常は直流には埋め込まない工夫がなされている。そこで、これらは異なる３Dとした。CampisiらはQIMだけでなくQIMとRDMの比較を行っている。RDMはランダム・ディザー・変調の略で、スケールの変化に強いという特徴を有す[3-7]。最近では、Fu等が、３次元DCTを提案し、基本特性を調べているがQIMは使用されていない[3-8]。変換した周波数成分に値を加える形で埋込みを行う方式である。

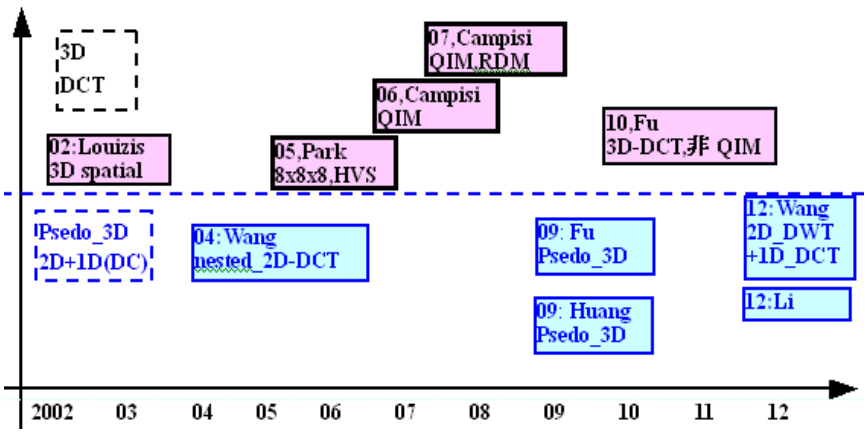


図 3-１　3D-変換を使用する電子透かし方式の比較.

本論文では、これらの例を踏まえて、変換として、第一に3D-DCTを用い、基本特性を調べる。1次元変換と2次元変換の埋込みに比べ、どの程度の差異があるか定量的に明らかにする。これを行うため、埋込みの数を揃える必要がある。例えば、1次元変換で長さNに対し、直流の隣に1個の埋込みを行うとする。同じ状況をN×Nの2次元変換ブロックで行う時、即ち直流の隣の水平成分1個、垂直成分1個、斜め成分1個の合計3個の埋込みを行うとする。すると１次元変換では、要素の数に対し、埋込み成分は、1/Nとなる。一方、2次元変換では、3/(N×N)となる。この二つ比率を共に埋込み個数に対応した整数にする工夫を要する。更に3次元まで一貫して揃える事も難しい。そこで、本論文では、以下、なるべく、埋込み個数が整数であることに対応するように、揃えると共に、揃えられない場合は、等価な変換を決めて公平な比較になされるようにしていく。

また、変換面での埋込みは、画像の値に大小の変化を引き起し、逆変換後に０より小さい値になる場合や、255を超過する場合がある。そのための対策と、係数精度が、このオーバーフローに与える影響とを調査していく。また変換には、第一にDCTを用いるが、Chirp変換[3-9]を用いる新しい方式の提案を行い、有効性を示していく。

3次元の場合は、埋め込みの構成をするために、以下ブロックのサイズをN=8とした場合について説明する。動画から8個のフレームを取り、画像データを8*8の画素ブロックに分割し、各フレームの画素ブロックを周波数成分に変換する。フレームの同じ位置にブロックを選ぶ。再び、そのブロックにDCTを施し、埋め込み情報を得ることができる（図3-2）。

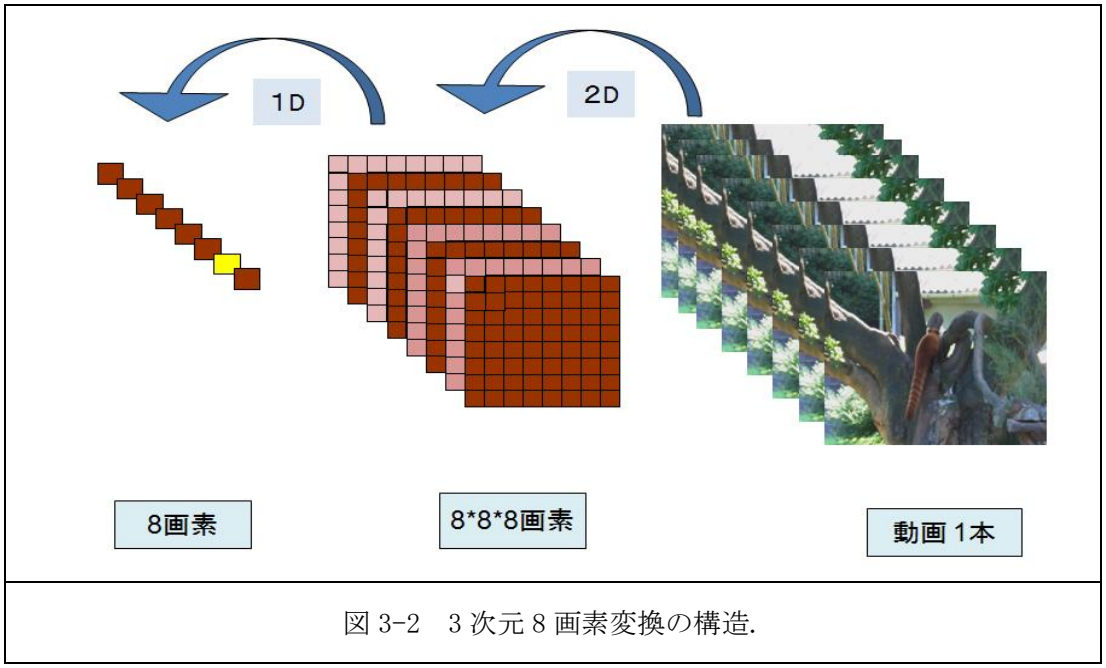


図 3−2　３次元８画素変換の構造.

３次元の DCT 係数を F(u、v、w)とすると、ブロックの大きさが N*N*N の３次元画像信号 f (x、y、z)に対する DCT および IDCT はそれぞれ次式によって表される。

$$F(u,v,w) = c(u)c(v)c(w)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\sum_{z=0}^{N-1} f(x,y,z)\cos\left[\frac{(2x+1)ux}{2N}\right]\cos\left[\frac{(2y+1)vy}{2N}\right]\cos\left[\frac{(2z+1)wz}{2N}\right] \quad (3\text{-}2\text{-}3)$$

$$c(u) = \begin{cases} \frac{1}{\sqrt{2}}, & u = 0 \\ 1, & u > 0 \end{cases} \quad c(v) = \begin{cases} \frac{1}{\sqrt{2}}, & v = 0 \\ 1, & v > 0 \end{cases} \quad c(w) = \begin{cases} \frac{1}{\sqrt{2}}, & w = 0 \\ 1, & w > 0 \end{cases} \quad (3\text{-}4)$$

また、N*N＊N 画素の IDCT の式は

$$f(x,y,z) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\sum_{z=0}^{N-1} c(u)c(v)c(w)F(u,v,w)\cos\left[\frac{(2x+1)ux}{2N}\right]\cos\left[\frac{(2y+1)vy}{2N}\right]\cos\left[\frac{(2z+1)wz}{2N}\right] \quad (3\text{-}2\text{-}4)$$

である。

DCT 後は画像に AC 周波数成分の無い平坦部では、変換後の係数の絶対値が小さく、電子透かしの埋込みが難しくなる。そこで、平坦部でも変換後の係数値が大きくなるように、非対称の変換が開発されている。

従来からある DCT による変換を使用したとき、4 次では、埋込み位置は、第二成分か第三成分かが候補になる。DC は他の係数より、影響が大きく、通常埋込みを行わない方が良いとされている[3-12]。 しかし、大ブロックの場合や 2 次元の場合は、埋込み位置は多数の候補があるが、4 次の場合は、選択肢が無い。DCT 係数は、定義により予め定まった離散的な周波数を表し、画像に適合した周波数や、埋込みに好都合な周波数位置を選ぶことができない。それならば、DCT という固定の変換を用いずに、第二成分位置に対して最も好適な周波数を有する変換を開発することが有効である。

DCT に対し、それとずれた周波数を生成する候補として、SLANT 変換[3-13]がある。SLANT 変換はその名の通り斜めに並んだ係数からなる。しかし、文献[3-13]の SLANT 変換は単に係数が斜めに並んでいるだけで、依然として対称な形式である。4 次の変換を電子透かしの埋込みに用いる場合の問題をより詳しく把握するため、テスト画像を用いて、アダマール変換[3-14]や DCT 変換を行う予備実験を行った。4 次の DCT を用いた変換後の値の分布は、図 3-5 左のようになっている。この分布は、8 次の JPEG における DCT の変換でも多数示されている[3-15]。画像によってまた、場所によって分布は異なり、平坦な部分では、DC に集中し、第二成分はほぼゼロになる。このような場合は、埋込みを避けているのが通常の例である[3-15]。DCT の変換後の AC 係数がゼロになるのは、DCT の変換係数が正負対称で均整がとれているからである。偶数個の画像データでブロックを形成する場合は、変換の係数を非均衡にすれば、これが打開できると予想できる。また、画像ブロックを奇数個、例えば 5 個にすれば、DCT であっても必然的に非対称になり均衡は崩れる。ここでは、従来 DCT との直接比較をするためにも、まず 4 次の偶数ブロックのまま、変換の斜交化を行うこととする。埋込み位置を画面全体から広く選べるようにするためには、このように

平坦部にも埋め込むことができることが望ましい。

斜交変換としては、画像の圧縮に用いた例がある[3-16]。斜交になり、画像のブロックが正方の形だけでなく、傾いた斜め線からなるブロックが使用可能になり、しかも斜め線のエッジに対する符号化特性が向上することが確かめられた。また、新規に変換全体を設計したため、第一成分も均一な係数でなく、曲がっているため、グラデーションのようなやや傾いた平坦部の特性が向上することなどが示されている。この斜交変換は、係数が非均衡であるため、上記の目的に一応適合している。しかし、非均衡であるうえに加えて、上記電子透かしの埋込みの目標は、画像の平坦部も非平坦部も、なるべく広い部分で、埋込みが可能になることである。変換は、入力ブロックの周波数と競合して、変換後の係数の絶対値が大きくなることが望ましい。即ち変換をある周波数に固定すれば、それに適合して、変換後の係数の絶対値が小さい現象が起こりうる。そこで、周波数は、ある固定の最適値があると想定して、それを求めるのではなく、むしろ、1 個の成分に対して、多数の周波数が混ざった変換を実行することがよいことが分かる。以上の考察の元に、Chirp 変換という漸次周波数が上昇していくような変換を導入することにする。Chirp 信号は、アナログ時間信号形式では、

$$C(t) = A \cdot \cos(2\pi f(t) \cdot t + \varphi) \qquad (3\text{-}2\text{-}5)$$

と表される。$f(t)$ が 1 次式のものを線形チャープ信号と呼ぶ。従来の DCT やフーリエ変換などは皆、1 個の基底が単一の周波数から成り、かつ対称性を有している。ここで目指す形状は周波数が複数にわたることと、全体の正負の割合が同じという対称性を有しないことである。このためには、Chirp 信号を正負の割合が一致しないように切り出せばよい。このような形状の波形を電子透かしの埋込みに使用した例として、文献[3-17]がある。DCT 後の係数を大きさに応じて徐々に緩やかに正負反転する Index 関数により求まる値を透かし値として埋込む。この Index 関数は、Chirp とは逆に次数が大きくなるほど周波数は低下する。また、２値に量子化されており、変換では無く、量子化して透かし情報の生成に使用される。また、文献[3-18]では、乱数から鍵に依存した直交変換を順次作り、変換として使用している。これは、これまでの DCT などの直交変換でなく、推定が難しいと思われる変換を使用している。しかし、変換は乱数を元に設計されており、画像の低域成分の考慮は、設計手順上なされていない。

DCT やフーリエ変換の様な低域重視の変換を考慮しつつ、多数の周波数を組み込もうとしている。また、小さいブロックに 1 カ所だけ埋め込むような場合の一箇所の埋込み位置に対応して、効率的な変換基底を設計する。以下 4 次と 8 次の場合の設計を示す。まず、埋込みを第二成分に行うものとして、第二成分を生成する基底を Chirp 変換によって実現する。離散的な 4 点又は 8 点で Chirp 信号を生成するため、DCT の項を参考に低次から中域の周波数を生成するように、係数を定める。高い周波数成分は、多くの場合、画像に含まれないため、高い成分を組み込んでも、活用されない。そこで最大でも DCT で言えば中間の周波数に当たる部分までを参考にして係数を決める。図 3-3 に 4 次と 8 次の Chirp 変換の係数例を示す。

8 次の場合を見れば、低域の半波長から中域まで周波数が変化していることが分かる。また、各基底の和は、DCT などのようにゼロになっていない。これが非均衡な直交変換の第二基底である。次に第一基底を作る。第一基底は通常平坦な直流であったが、必ずしも完全な平坦でなくても直交変換を構成できる。実際画像データから 2 乗誤差最小となる所謂最適な変換、KL 変換を設計すれば、第一成分はかすかに曲がった物になることが多く、完全に平坦な直流になるとは限らない。また、第二成分の形状を優先して先に決定するため、その係数の合計値が非ゼロであることから、平坦な直流では、直交しないことになる。第一成分は、第二成分と直交する係数で、緩やかな平坦になるように決める。次に第三成分以下を決める。ここで、今回の目的は、画像圧縮などではないので、実際に透かし埋込みに使用する第二成分のみが透かしの埋込みに直接の影響を持つものであり、他の成分は影響度が小さいので、DCT の第三次以降の係数を種として使用し、Gram-Schumidt の直交化法で正規直交系列に組み上げる。

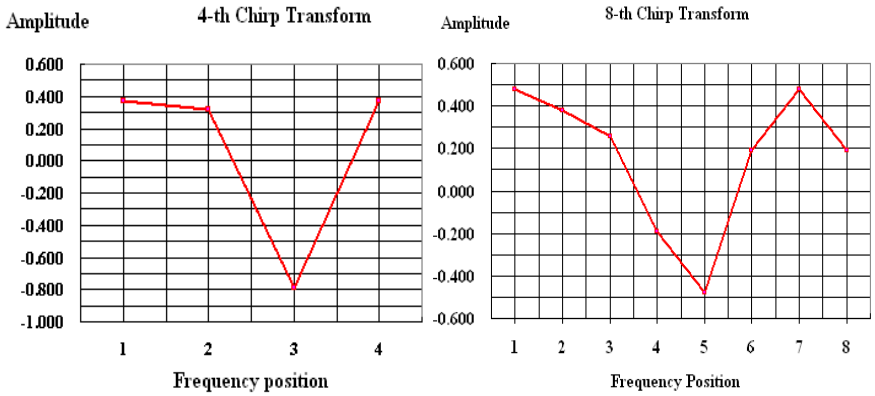
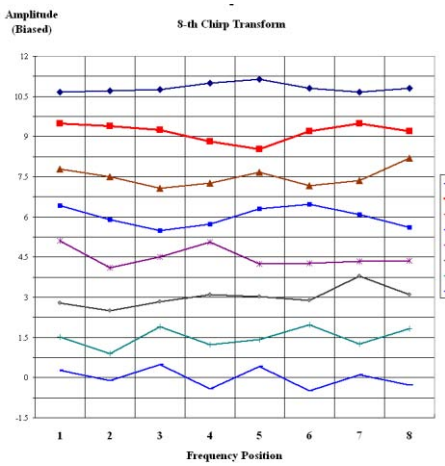


図 3-3　　4 次と 8 次の Chirp 変換の係数例　波形図.

第一成分を適当に作った場合も、Gram-Schumidt の直交化法で、正規化しておく。第三成分は、DCT の第三成分を仮に当て、Gram-Shmidt の直交化法で、正規直交化していく。以下同様に、DCT の係数を借りて、正規直交基底を作り上げる。図 3-4 にその形状を示す。第二成分以外は、変換に使用されるが、量子化等の変形を受けることなく、透かしを埋込んだ第二成分と共に逆変換される。従って、省略はできないのと、逆変換の時の誤差感度への影響を持つという必要性はあるが、この目的においては、第二成分のみが最も影響が強いので、第二成分の設計に最も注意を払えばよい。また、実際の変換として使用する場合に、係数の絶対値はゼロに近いものは変換において寄与率が小さくなり、情報が無駄になる。そこで、係数の絶対値はなるべく大きい数値になるものが採用されるよう、調整してある。

図 3-5 右にこの変換を用いた時の変換後の係数の分布の様子を示す。先の図 3-5 左と比べると、明らかに分布が右に広がっていることと、それに伴い、密度が薄くなり、良く分散している様子が分かる。実際、空のような平坦部も、かすかな変動が有るため、それに呼応して、分布の中心が移動しただけでなく、分散も広がっていることが分かる。全体とし

て変換後は、非ゼロの係数が大幅に減少し、埋込みが画面のより広い部分で可能になることが分かる。



$$\begin{bmatrix} 0.151 & .200 & .265 & .497 & .646 & .299 & .151 & .299 \\ .476 & .381 & .257 & -.190 & -.476 & .190 & .476 & .190 \\ .299 & -.002 & -.439 & -.242 & .163 & -.345 & -.137 & .704 \\ .416 & -.103 & -.502 & -.277 & .295 & .482 & .089 & -.401 \\ .603 & -.403 & .001 & .565 & -.244 & -.225 & -.150 & .146 \\ -.225 & -.504 & -.170 & .086 & .034 & -.121 & .785 & .105 \\ .017 & -.614 & .388 & -.279 & -.073 & .467 & -.242 & .336 \\ .271 & -.104 & .490 & -.420 & .420 & -.490 & .104 & -.271 \end{bmatrix}$$

図 3-4：8 次 Chirp 変換波形図と係数

その変換は、Chirp 変換といい[3-3]、単一ではない周波数を一つの変換基底に含む。

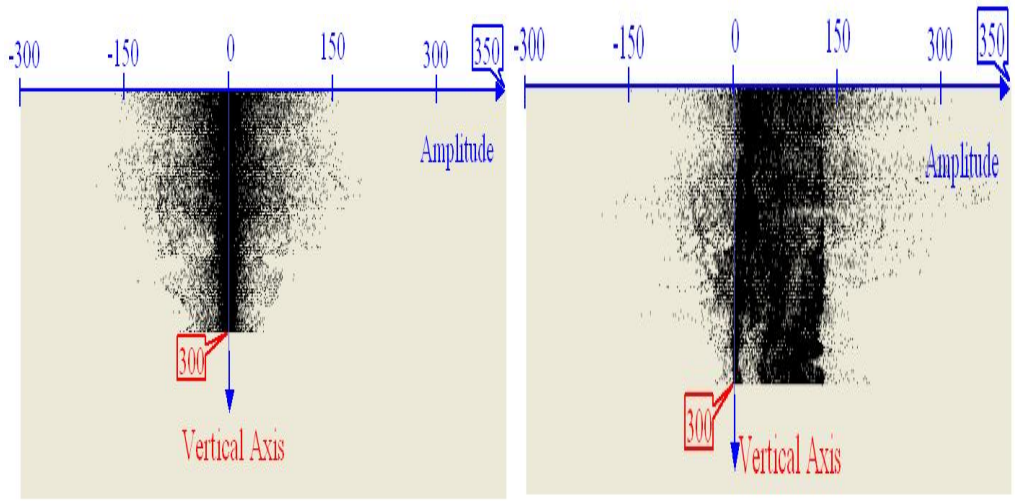


図 3-5　8 次 Chirp 変換後の第二係数の分布[9]右、
DCT 変換後の第二係数の分布[9]左.

図 3-5 右にこの変換を用いた時の変換後の係数の分布の様子を示す[3-9]。画像の真っ白のような平坦部も、非対称性により、分布の中心が移動しただけでなく、分散も広がって

いることが分かる。変換後は、非ゼロの係数が大幅に増加し、埋込みがより広い部分で可能になることが分かる。

以下、1 次元から 3 次元の変換の仕様と埋込み位置について具体例によって説明する。

3-2-1　1 次元変換の場合

埋込み位置に関し、サイズ 720×480 の静止画像を用意する。 NTSC(National Television Standard Committee) が用いている表色系は、Y (輝度) I(オレンジ-シアン) Q(緑-マゼンダ) の三要素からなる。次にこの YIQ 変換をする。 Y、Ｉ、Q は色変換で式(3-2-6)により、変換される。Y 成分を 1 ブロックが 8 画素から成る 1 次元ブロックを選択した後、 各ブロックに 1 次元 DCT を行う。埋込みブロックの埋め込み対象要素に後述する区間特性を有する QIM を施して情報を埋め込む。埋め込み位置である低中域成分を利用する。DCT 処理後の DCT 係数は、第一個目（F[0]）が直流成分となり、その他の F[u]が交流成分となる。埋込みは 2 個目の成分（F[1])に対して行う。QIM の量子化の基準幅は 4 から 128 等である。QIM の基準幅は大きくなれば、大きいほど画像の劣化も大きくなる。逆に QIM の基準幅は小さくなると耐性が劣り、検出が難しくなる。各ブロックに逆 DCT を行い、埋め込み後の画像を得る。埋め込む対象となる 6 個のブロックにおいて、各 1 ビットの埋込みが行われ、合計 6 ビットの情報を埋め込む。

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (3\text{-}2\text{-}6)$$

3-2-2　2 次元変換の場合

Y 成分を 1 ブロックが 8×8 画素から成る 2 次元ブロックを選択した後、 各ブロックに 2 次元 DCT を行う。埋込みブロックの埋め込み対象要素に QIM を施して情報を埋め込む。埋め込み位置である低中域成分として、第 2 番目の位置を用いる。DCT 処理後の DCT 係数の場合、第一個目（F[0、0]）が直流成分となり、その他の F[u、v]が交流成分となる。埋込みは 2 個目の成分が含まれる（F[0、1]、 F[1、1])、（F[1、0])の 3 個に対して行う。QIM の量子化の基準幅は 4 から 256 等である。逆に QIM の基準幅は小さくなると検出も難しくなる。各ブロックに逆 DCT を行い、埋め込み後の画像を得る。埋め込む対象となる各ブロックが 2 枚の画像に渡り適用され、各 1 ビットの埋込みが行われるため、合計 6 ビットの情報を埋め込む。

3-2-3　3 次元変換の場合

3 次元ＤＣＴの場合は、８枚連続の静止画像を利用する。上記 2 次元の説明の通りで、先ず、各 8 枚の画像を 2 次元 DCT する。DCT の次数を 8×8 に設定して、64 画素を 1 つの 2 次元ブロックとし、周波数領域へ変換する。次に最後の 1 次元変換を各 64 個の変換後の値に対して行う。埋め込み位置である低中域成分として、1、2 次元の場合に合わせて第 2 番目の位置を用いる。 DCT 係数の第一個目（F[0、0、0]）が直流成分となり、その他の F[u、v、w]が交流成分となる。埋込みは 2 個目の成分が含まれる（F[0、1、0]、 F[1、1、0])、（F[1、0、0]、 F[0、1、1]、 F[1、1、1])、 （F[1、0、1])の 6 個に対して行う。これ以外で第二番目を含む 3 次元変換後の位置として、F[0、0、1]があるが、合計で 7 個という素数なり、あとで行う次元間の整合性が悪い場合が出るので、偶数の 6 個とした。

３−３　埋込み・検出・耐性評価実験

3-3-1　原画像の検出

1、2、3 次元の DCT 処理後の埋込み処理を行い、JPEG 圧縮による耐性実験を行った。次元による比較と Chirp 変換の特性評価も行っている。

画像は、動画像から切り出した連続する静止画像 8 枚を用いた。図 3-6 に実験に使用した画像の例を示す。

図 3-7 は埋込みを行わない原画像の検出率であり、横軸は量子化の基準幅であり、縦軸は検出率である。埋込み数が少ないので、ばらついているが、50%を中心とする分布になっている。

図 3-6　　原画像(ppm)の 1 フレーム、720×480 画素.

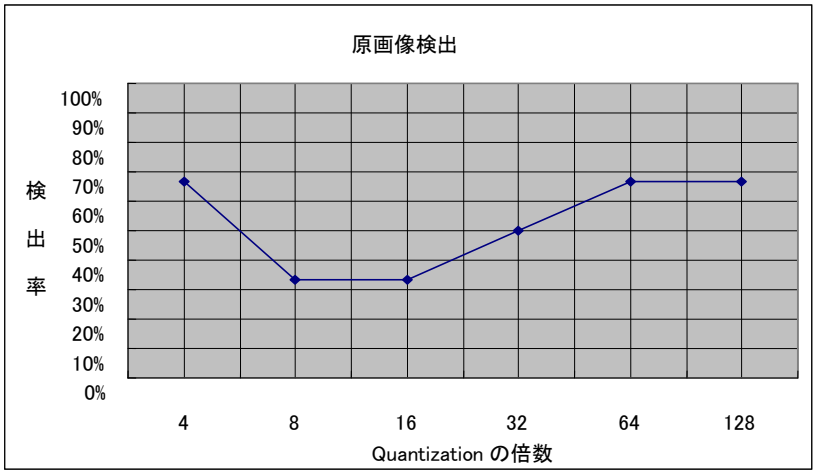


図 3-7　3 次元電子透かしの耐性：原画像検出率.

3-3-2　DCT 埋込みの場合

検出の判定の方式について、埋込みと同一の変換を行い、その値との関係で量子化区間を割り当てることで、埋め込まれた電子透かし情報を検出する。量子化の基準幅が 16 である時、係数値の値は-8≦係数値の値≦8 に対応するビットを「透かしがあり」と見なし、-16<係数値の値<-8 または 8<係数値の値<16 に対応する値を「透かしが無し」と見なす。図 3-6 に示してある。

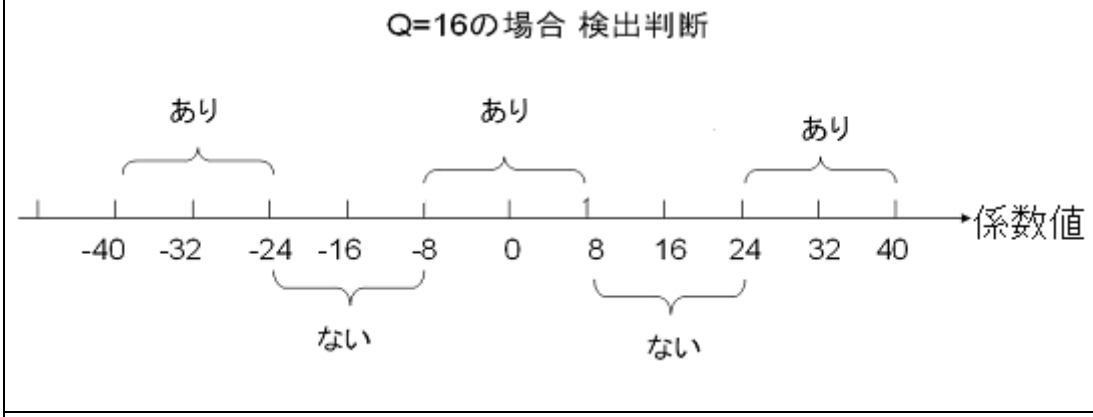


図 3-8　検出判定区間.

DCT を用いて電子透かしの埋込み処理について、1 次元 DCT の場合は DCT の次数を 8 に設定して、周波数領域へ変換する。 8 個の要素に対して、低周波数に相当する 2 個目を所

定の特性で量子化器 Q で量子化し、IDCT で逆変換され、画像に戻す。

埋込み位置と透かしの個数に関し 1、2、3 次元で埋込みビットは整数であるため、構造上変換後の埋込み位置が異なってくる。本研究では、動画像の 3 次元データへの埋込みに対し、原画像でほぼ同一の位置にある 1、2 次元のブロックを用い、整合性を取るようにした。

図 3-9 に JPEG 圧縮による電子透かしデータの耐性を示す。量子化の基準幅は 32 である。縦軸は検出率の判定を示してある。横軸は JPEG 圧縮時の画像の圧縮データ容量(kb：キロバイト)である。6 カ所のブロックに 6 個電子透かしを埋め込んだ。例えば、1 /19(53/1013)でも 83.3%であるが、結果として埋め込む位置は 1 ブロックに 1 個しかないため、圧縮の攻撃が強いと、検出率が低くなる。

注:横軸の 21、73 など圧縮したデータサイズ（単位 kb）である。1/19 の意味は 1÷（1013÷53）のことである。

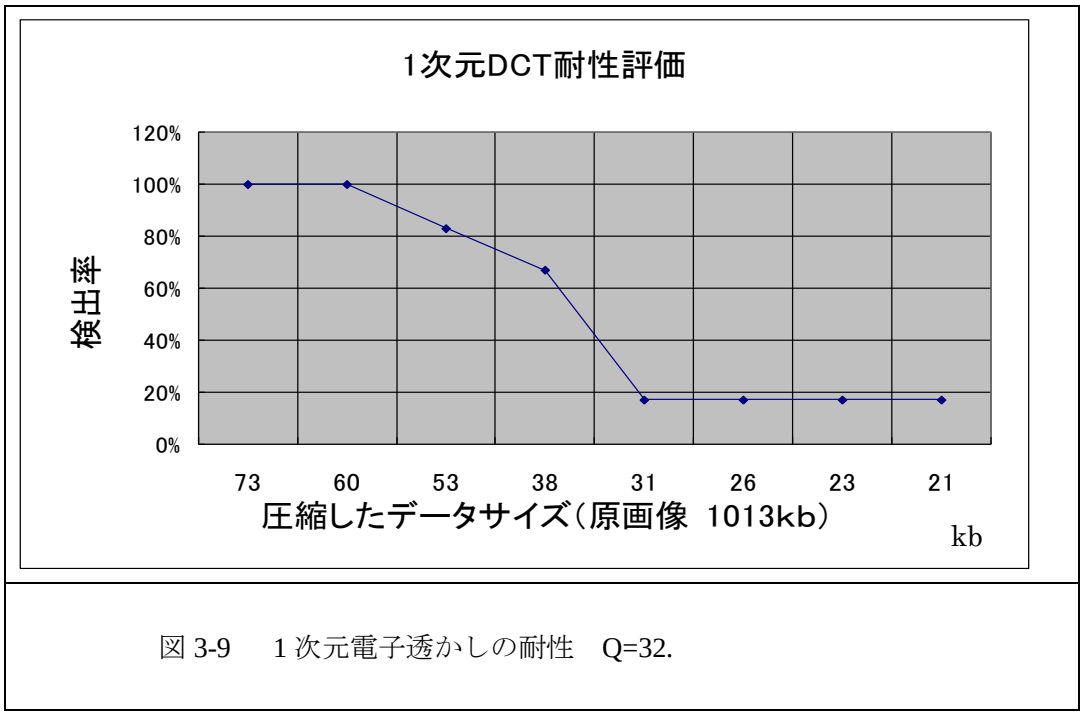


図 3-9　1 次元電子透かしの耐性　Q=32.

2 次元 DCT の場合は 8*8（64）画素を 1 ブロックとする。8*8 画素のブロックを変換して、低周波数に相当する 2 個目を含む変換後の係数値を所定の特性で量子化器で量子化する。

圧縮率が 1 /19 までは透かし情報が 100%残っている。埋め込みは 2 枚の静止画に 6 個、1 枚あたり 3 個の透かしである。3 個分が、2 次元の 8*8 画素に分散されている。一方、1 次元の場合は、同じ 3 個の透かしは 24 画素に埋められている。1 次元と 2 次元を比較すれば、原画の画素数は 1 個の透かしに対し、各 8 画素と 64/3=21.33 画素となり、2 次元の方がより広い領域に拡散される。このことにより、圧縮の攻撃に対し、1 次元より耐性が強くな

ったと考えられる。（図 3-10）

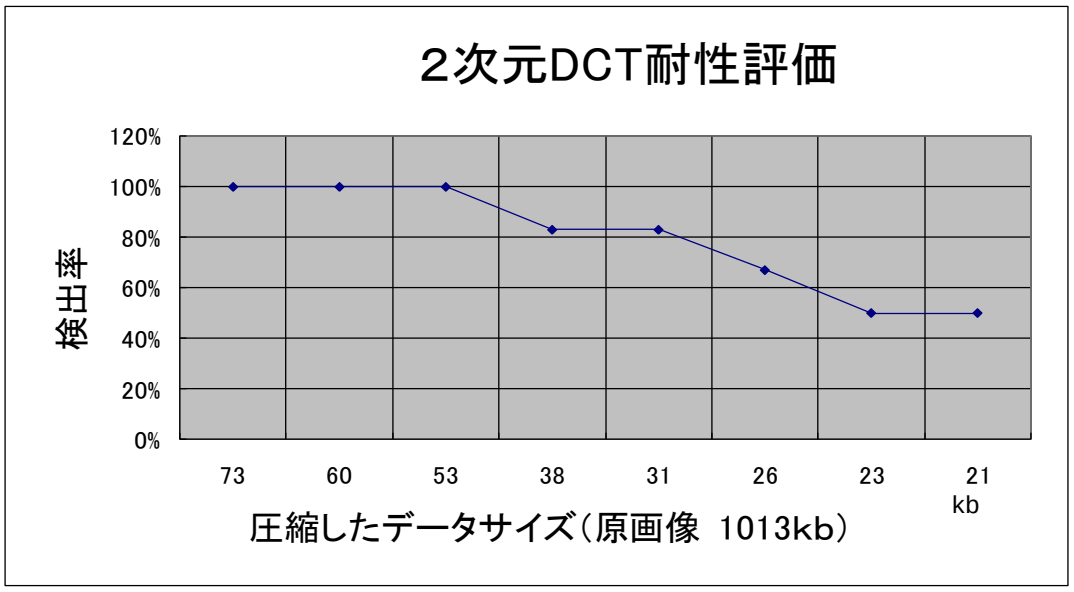


図 3-10　２次元電子透かしの耐性 Q=32.

３次元の場合は、動画として８枚の画像に埋込みがなされる。量子化の基準幅は Q=32 である。縦軸は攻撃実験の段階での検出の判定を示してある。横軸は画像の圧縮後の容量(kb)である。圧縮率耐性は 1 /32 まで 83.3%。埋め込む位置は３次元の 8×8×8=512 画素のブロックに 6 個埋められるため、1 個の透かしに対し 512/6=83.33 画素の原画が対応しており、圧縮の攻撃により、1、2 次元より耐性が強くなると考えられる。（図 3-11）１次元は最も性能が悪い結果となった。

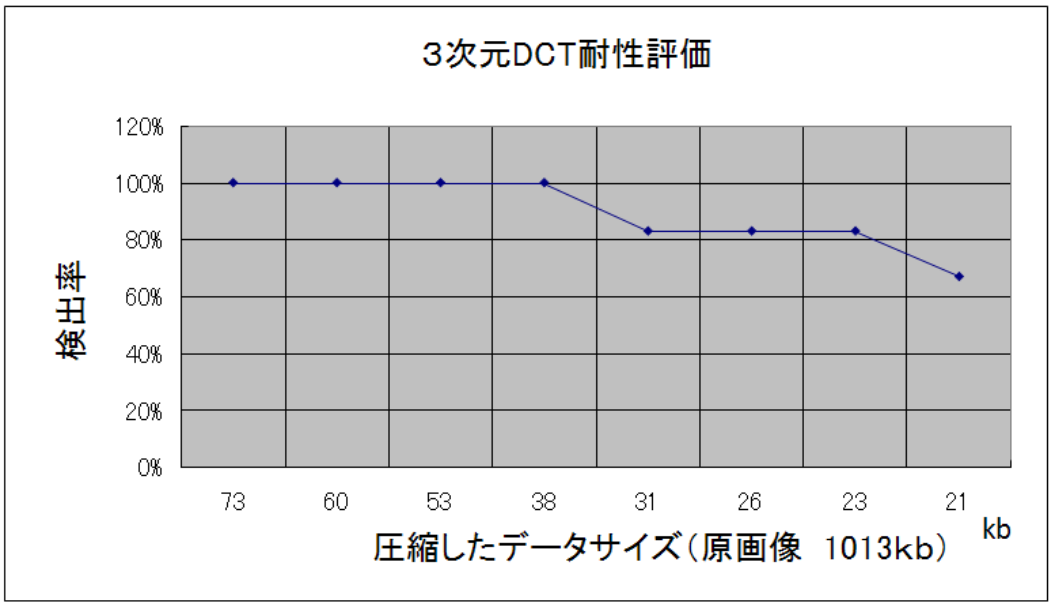


図 3-11　3 次元電子透かしの耐性 Q=32.

ここまでの実験で、1 次元より、2 次元、2 次元より 3 次元が耐性に関し効率が向上する。即ち、1 個の透かしに対する埋込み画素数が多いため、耐性が高くなると考えられる。埋込みに伴う劣化を S/N 値で示してある。S/N の計算は、(3-2-7)式を用い、S として信号のピーク値(S=255)、N は埋込み後の画像値と原画像の差の平均を用いている。誤差は、埋込みブロックにしか発生しないが、S/N 値は画像全体に対して計算してある。埋込んだ透かしの個数が少なく、S/N 値はかなり大きいが、参考文献[3-10]で検討したように、次元により個数が変化すると、事後に個数と S/N 値を調整して、比較することが必要になるため、ここでは、周波数領域でほぼ同一の 2 番目の要素を中心にして、個数 6 個で統一した。埋込み個数を増やす場合は、ブロック単位で、2 倍増加する度に、平均２乗誤差(MSE）が２乗前の信号で $\sqrt{2}$ になるため、3dB 減じれば平均的には近似できると考えられる。

$$SNR = 20\log_{10}\left(\frac{S_{pp}}{N_{rms}}\right) \quad [dB] \qquad (3\text{-}2\text{-}7)$$

埋込み個数を 6 個に統一することにより、次元により、画像のフレームの使用範囲が異なってくる。1 次元の場合は、1 枚、2 次元の場合は 2 枚のフレームを使用しているので、S/N 値の直接的な計算結果は表 1 の様になっている。埋込み位置はブロックの起点を(199、199)に揃えた。1 次元については、計測だけ 2 枚の画像で行っている。これを 8 フレームの範囲まで拡大し、換算したのが、表 3-1 である。確認のため、他の枚数での結果を追試してあるが、(3-2-7)式の計算において、枚数が倍増するときには、Nrms は $1/\sqrt{2}$ になり、S/N 値

で約 3.0dB 増加する。そこで、以下 3dB ずつ加算することにより換算することにしている。

表 3-1　埋込みビット数と S/N、Q=32.
誤差の計算範囲は 1 次元では、8x1 の変換を 6 カ所、２次元は、8x8 のブロックへの埋込みを画像 2 枚について、3 次元は 8x8x8 の 8 枚の画像での計測。埋込み位置は(199、199)を起点としている。

	1 次元	2 次元	3 次元
透かしの個数 （ビット数）	6	6	6
S/N　[dB]	78.5 (2 枚)	77.6 （2 枚）	83.4 （8 枚）

表 3-2 では、この変化が観察できるが、埋込みは、量子化レベルを Q=32 に統一し、６個の透かしによる誤差はほぼ同一と見做した場合の比較になっている。表 3-2 の最下段の差は逆変換の正規化係数の差異による影響と考えられる。1 次元と 2 次元では 0.9dB の差があり、2 次元と 3 次元では 0.2dB の差がある。この差のうち 1 次元と 2 次元は約 1dB あるので、後述する特性の評価で、1dB 分の差異を考慮していくことにする。ここではまず、埋込みレベルを調整し、その耐性を比較していくことにする。なおこれらの S/N 値は 70dB 以上ときわめて高いが、埋め込み１ブロックが（８×８）と小さいためである。基本性能比較するため画像１枚当たり（８×８の１ブロック）を基準に比較している。又、埋め込みによる劣化は小さいので、この埋め込みを多数の場所に増やすことができることも示している。

表 3-2　計測枚数に対する換算を行った後の埋込みによる誤差（S/N 値　[dB]）。
Q=32.

計測枚数	1 次元	2 次元	3 次元
1	75.5	-	-
2	78.5	77.6	-
4	81.5	80.6	-
8	84.5	83.6	83.4

３－４　Chirp 変換の場合

図 3-12、13、14 に変換行列を Chirp 変換に変更した場合を示す。耐性に関し、1 次元に対し、２次元が高くなっている。しかし、３次元では、また 1 次元程度に低下している。この原因は、Chirp 変換が、非対称で、かつ周波数が増加する係数を持ち、2 次元変換までは、その性質が保持されるが、3 次元目に於いては、変動が大きくなっているため、更なる変換で値が分散される傾向にあるためと思われる。

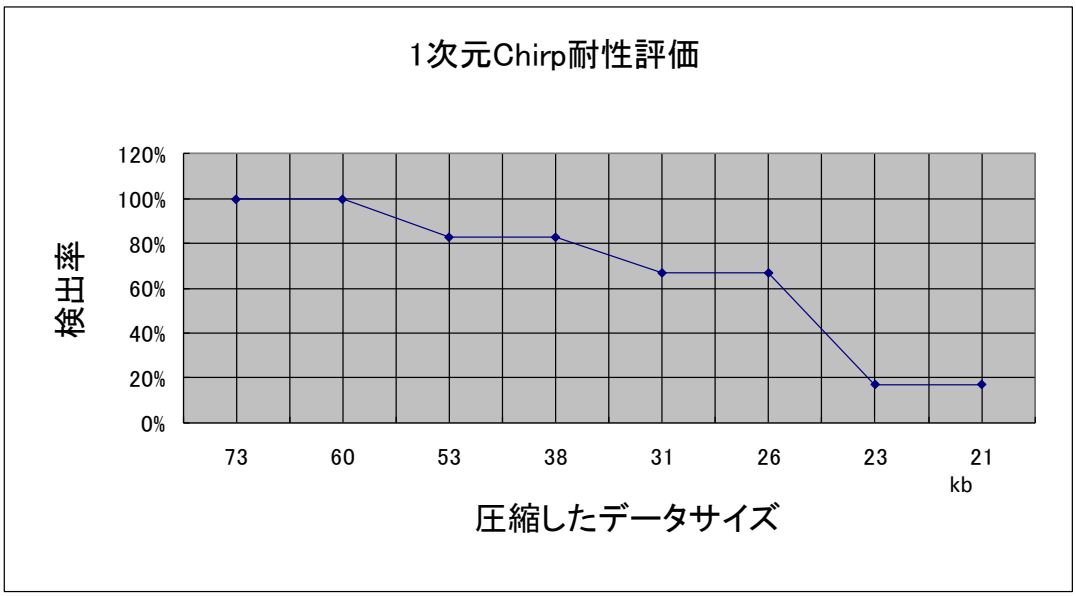


図 3-12 Chirp 1 次元電子透かしの耐性 : 埋込みの先頭の画素位置(199、199)Q=32、JPEG 圧縮.

本研究で開発してきた Chirp 変換[3-9]は 1 次元の基本構成であり、2 次元の場合は 2 サンプルからなる簡易型で、その形状は 2 次の Hadamard 変換になっている。すなわち、文献[3-9]の Chirp 変換は 2 次元以上に適用できるように作っていなかった。そこで、全てを Chirp 変換しないで、最終段だけ Chirp 変換を行い、その前までは、DCT で行う方法で、変換を行った。その結果を図 3-15、16 に示す。図 3-15、16 から DCT 後に 1 次元の Chirp 変換を行うと、耐性は、約 1/26 の JPEG 圧縮まで拡大している。Chirp 変換の効果により、性能が向上したと考えられる。

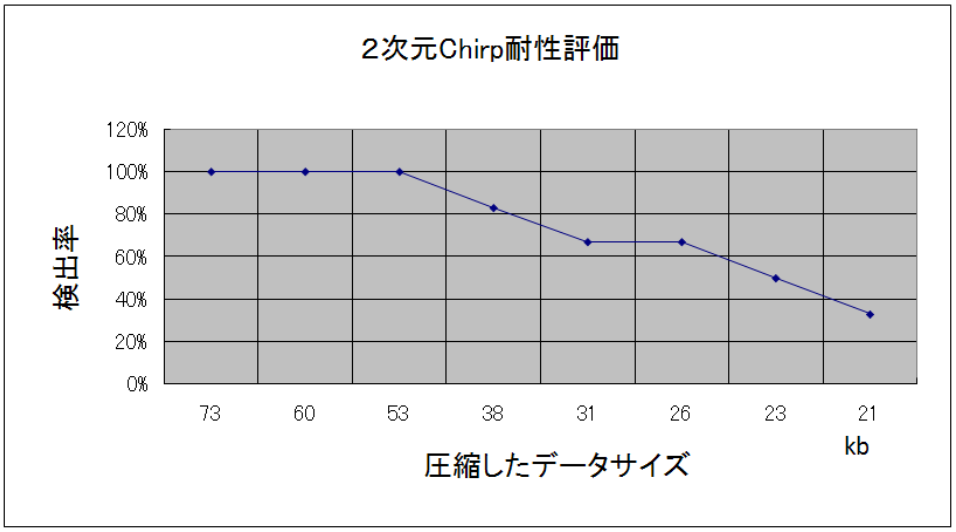


図 3-13　Chirp 2 次元電子透かしの耐性：埋込みの先頭の画素位置(199、199)Q=32、JPEG 圧縮.

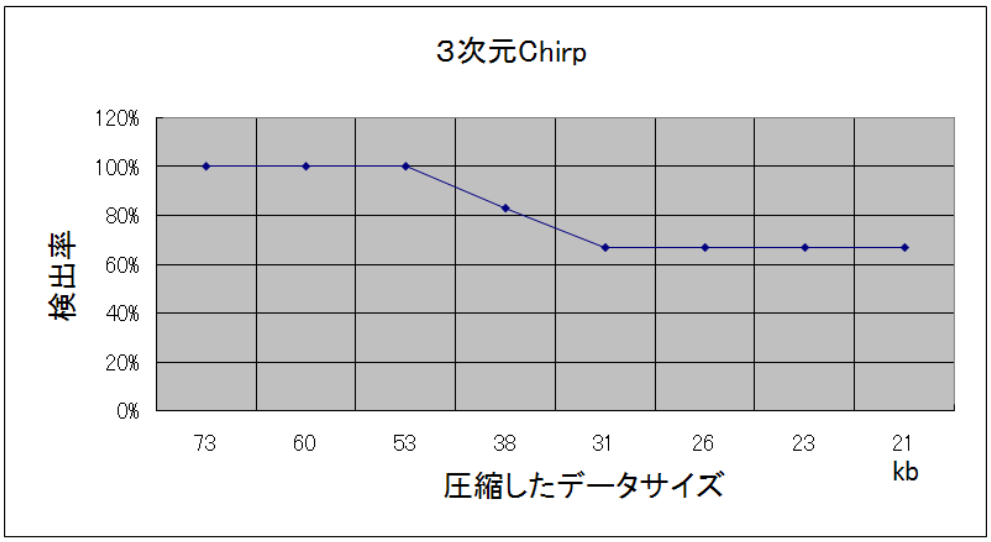


図 3-14　Chirp 3 次元電子透かしの耐性:埋込みの先頭の画素位置(199、199)Q=32、JPEG 圧縮.

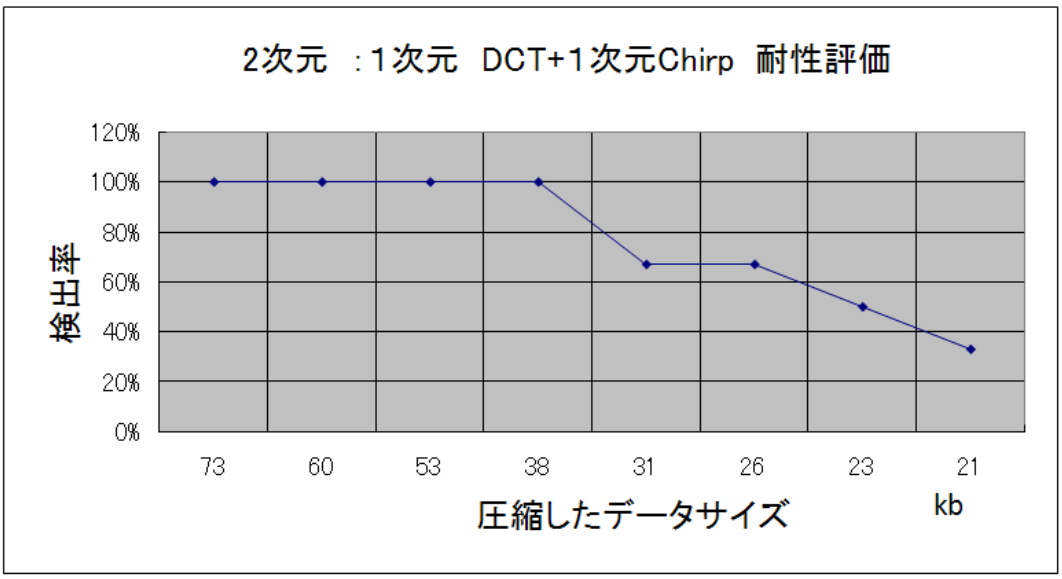


図 3-15 DCT+Chirp 2 次元電子透かしの耐性：埋込みの先頭の画素位置(199、199)Q=32、JPEG 圧縮.

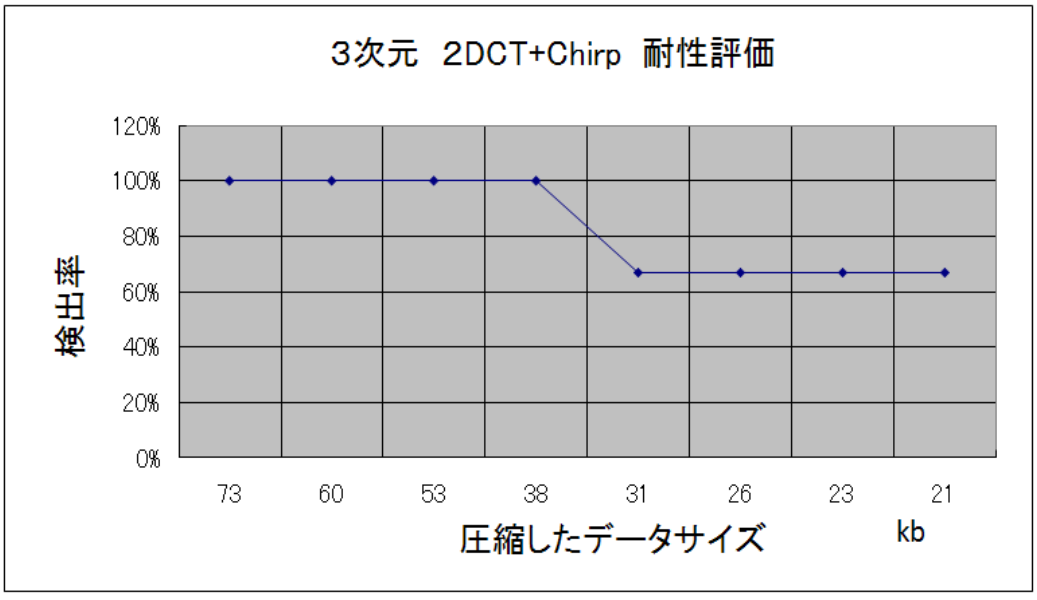


図 3-16 DCT+Chirp 3 次元電子透かしの耐性：埋込みの先頭の画素位置(199、199)Q=32、JPEG 圧縮.

変換自体は、直交変換として DCT と同程度の性能であると考えられる。埋込み場所に関しては、DCT は平坦部を避けて行う必要があるが、Chirp 変換はその必要がないというメリットがあり、有効な電子透かしの埋め込み位置が増加するという効果がある。

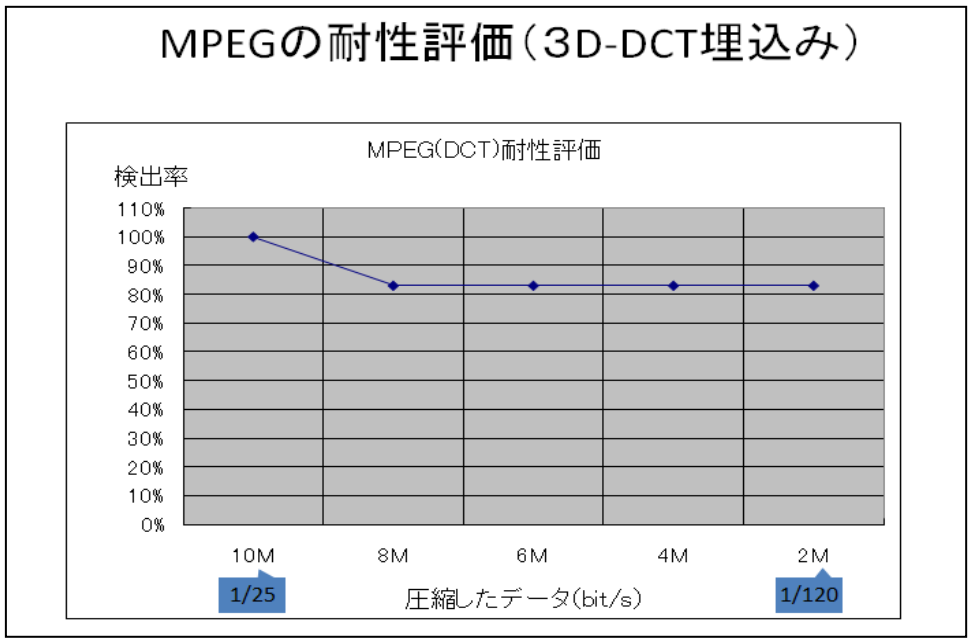
３－５　MPEG 圧縮に対する耐性

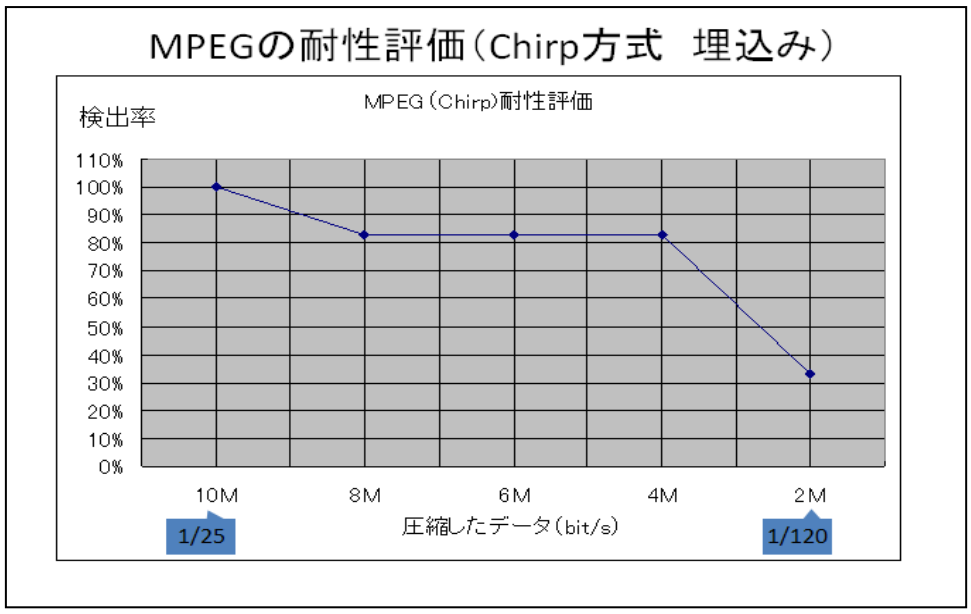
これまで、JPEG 圧縮に対する耐性に対する実験をしてきた。１、２次元の変換による埋込みは静止画像に対する方式であり、統一の基準で比較するため、３次元変換も JPEG 圧縮に対する耐性を評価してきた。しかし、３次元の画像データは動画像なので、動画像の圧縮方式である MPEG-2 方式による圧縮に対する耐性の評価を行った。

３D-DCT による埋込みの場合と、3D-Chirp 変換による埋込みを行う場合について、MPEG-2 圧縮を５種類のビットレートに設定して適用した。MPEG-2 圧縮は、MPEG 標準化で公開されている圧縮ソフトウェアを用いた。表 3-3 に実験仕様を示す。

表 3-3　MPEG-2 圧縮による耐性実験の仕様.

項目	仕様
圧縮方式	MPEG-2
使用ソフトウェア	mpeg2vidcodecv12.tar.gz (MPEG.MSSG)
実験環境	PC, Linux
入力画像	透かし埋込み後の ppm 形式連番ファイル８枚 720x480 画素
圧縮率	2,4,6,8,10　メガビット/秒（Mbps）
IBP 形式	M=2　GOP=8
音声	なし
圧縮後の形式	m2v
出力画像	ppm 形式連番ファイル８枚 720x480 画素





３－６　係数精度の影響

DCT、RGB、から YCbCr などの変換係数は桁数が増えれば、増えるほど正確な結果になる。実際には、RGB、YCbCr の変換と逆変換での誤差で若干画像の劣化がでていることがわかる。画像により視覚的に差異を観察するため、以下の図 3-17、18 に原画と埋め込み画

像の差を画像として示している。またその一部を画素値として示している。

10 進数で 5 桁以下のわずかな差異でも、透かし判定のスレッショルドの付近では、影響が変わりうるため、わずかではあるが、影響があることが分かる。計算としては、高精度で行う方がよい。下の図に示した例では、10 進で 5 桁を使う時の差の SNR は 51.16dB、10 進で 10 桁を使う時の差の SNR は 52.20dB。であり、視覚上の影響はないと言える。SNR は通常 35dB 程度が認識できる境界値であり、40dB 以上では、ほとんど知覚されることはないと言われている。従って、上に示した 50dB 前後の誤差は、知覚されないと考えられ、用いた 10 進数 5 桁の精度は十分高いと考えられる。

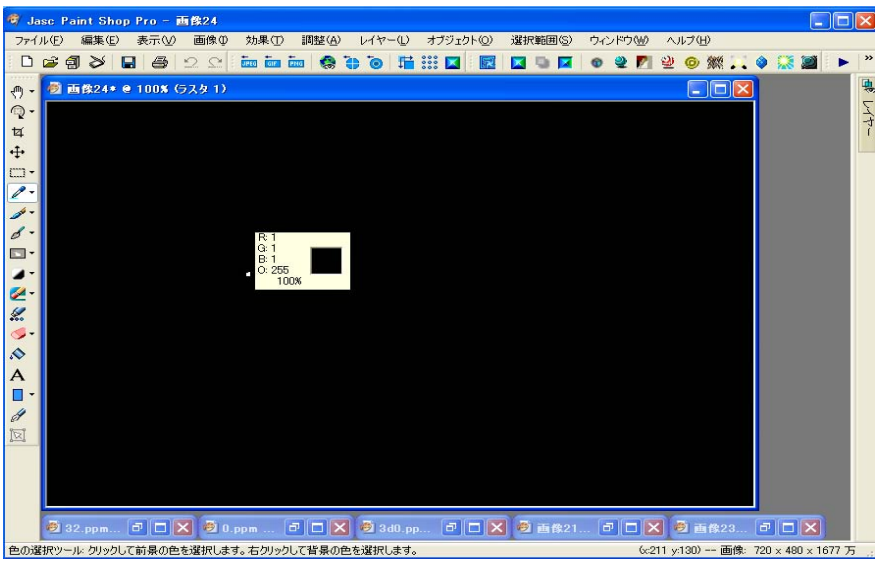


図 3-17　原画像と埋込み後の画像の差：変換は通常係数（10 進で 5 桁）で差は RGB(1、1、1)である.

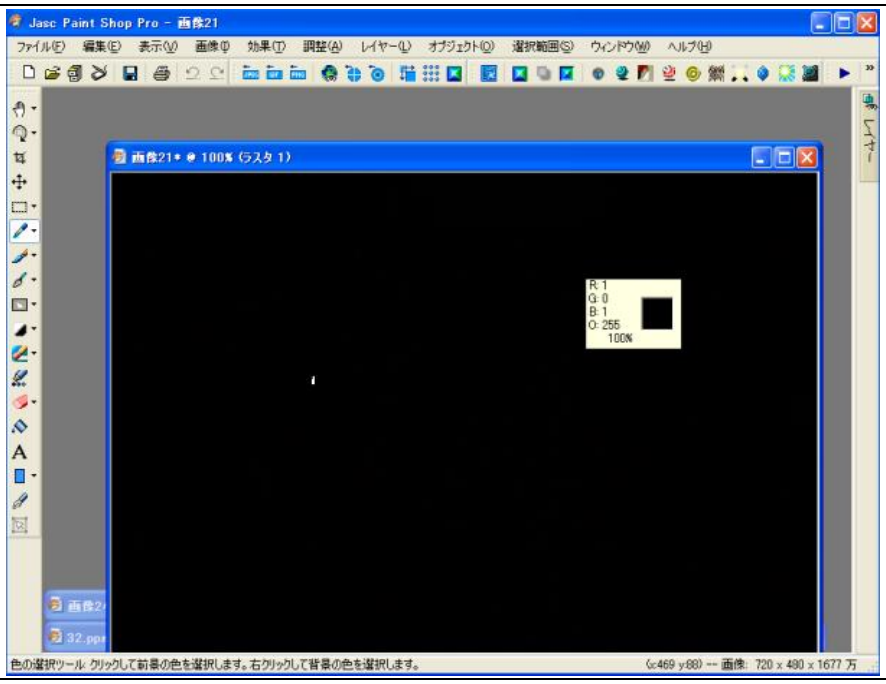


図 3-18　原画と埋込み後の画像の差。変換は高精度係数（10 進で 10 桁）で差は RGB(1、0、1).

３－７　他の画像での実験

実験は、他の画像でも行い、結果にはある変動があるが、一定の範囲内であり、同様の特性が得られている。DCT では、変換後の AC 係数値の絶対値が量子化レベルよりも大きくないと、埋込み前も後も共に 0 になり、検出は成功するが、埋込んだ透かしが 0 になって意味のない埋込みになってしまう。画像の場所により、この状態が変化しているので、多数の画像、多数の位置で埋込みを行い、その変動を調べておく必要がある。一方、Chirp 変換[3-9]では、DCT などの周波数解析で AC 成分が 0 になるところでも、非 0 の値になり、画像の位置によらず有効な埋込みがなされるという特徴がある。DCT に関し、図 3-19(a)～(b)に示す画像に対し実験を行った。結果を図 3-20、21、22 に示す。

図 3-19(a)　実験画像 2 演奏.

図 3-19(b)　実験画像 3 列車.

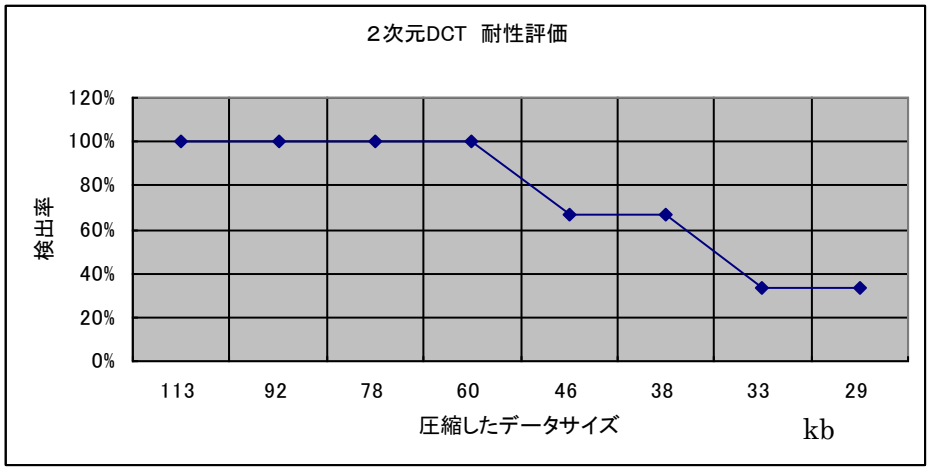


図 3-20　2D-DCT の実験結果、画像 2 演奏.

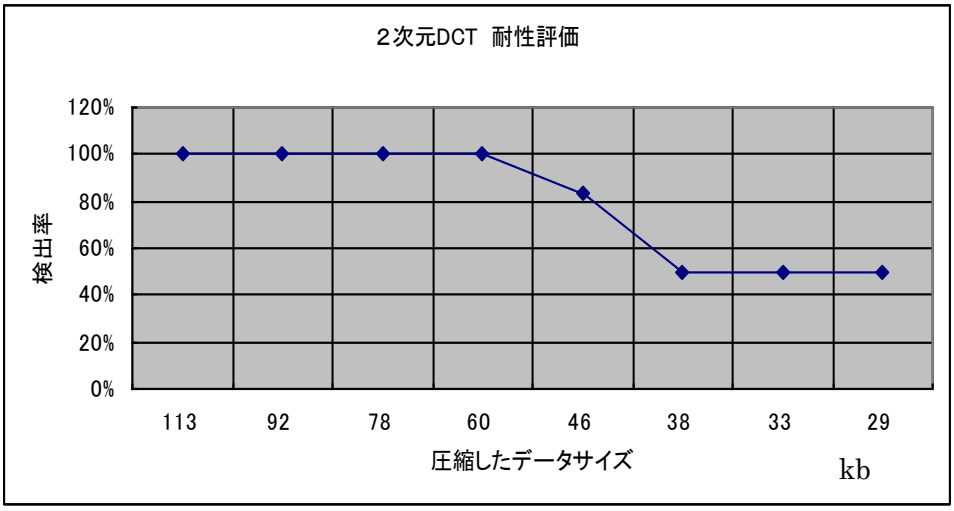


図 3-21　2D-DCT の実験結果、実験画像 3 列車.

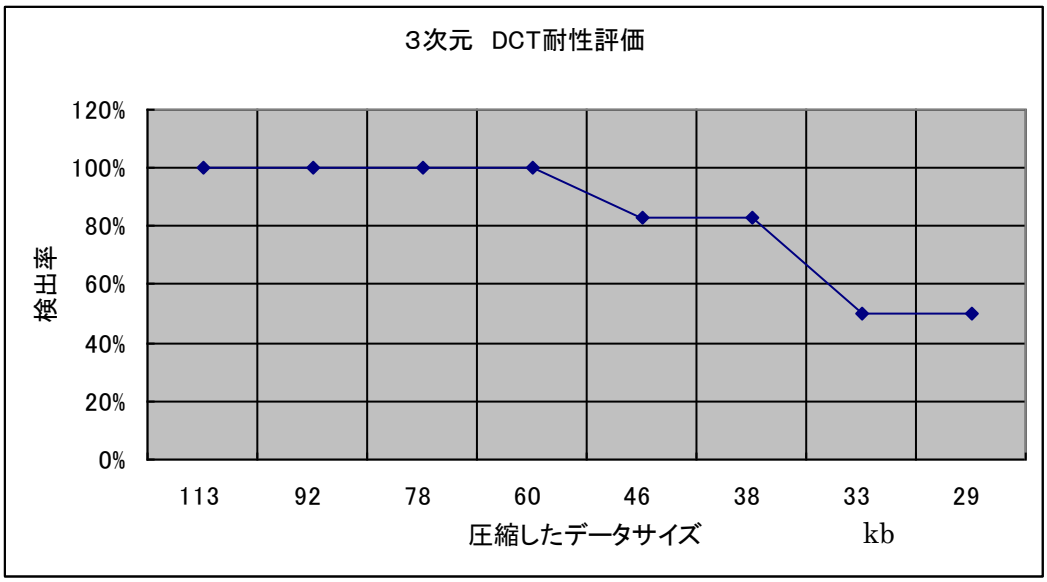


図 3-22　3D-DCT の実験結果、実験画像 2 演奏.

3－8　　考察

1 次元 DCT 方式について、電子透かしの検出率が特に低く、耐性が低いことがわかる。表 3-2 の換算後の SNR 値では、1 次元の方が、0.9dB 高いが、JPEG 圧縮耐性は圧縮率が 1/11 程度までしかなく、他の次元に比べても性能が低いと考えられる。 2 次元 DCT 方式について、電子透かしの検出率が 1 次元 DCT 方式より向上する。3 次元 DCT 方式について、電子透かしの検出率が 2 次元 DCT 方式より若干向上する。DCT 方式には、係数の絶対値はゼロに近いものは電子透かしとして寄与率が小さくなり、情報が無駄になる。3 次元の DCT と Chirp 変換を組み合わせる方式で、平坦部での埋込み機会が増大し、埋込みが画面のより広い部分で可能になることが分かる。Chirp 変換は 1 次元変換専用に開発されているため、又バイアス成分も付加されているため、2 次元以上の変換では、非対称のずれが増加し過ぎると考えられる。

透かし情報は画像により、JPEG 圧縮の攻撃に耐性が違う。そこで何枚かの画像で、埋め込み実験を行って来たが、検出率の変動は一定の範囲内にあり、特性評価を行うことができた。量子化の程度により、SNR が違うが、1 次元では量子化の基準幅 Q＝64 を境として、それ以上の Q 値では差の SNR は同じであった。これは変換後の振幅値が小さいためと考えられる。係数精度では、高精度係数で計算すれば、誤差も低くなるが、5 桁程度とれば、時差以上の画像レベルの変動の影響は小さく無視できる程度であるが、誤差は存在する場合があるので、注意する必要がある。SNR 値の比較では、一定の基準で換算を行い、埋込み個数の変動や、量子化ステップサイズの変化による差異を比較可能となるようにした。

３－９　変換による埋込み方式まとめ

変換に DCT を用いて、１、２、３次元変換後に量子化による埋込みを行う電子透かし方式を比較した。従来単独の結果しか得られていなかったが、埋込みビット数を揃え比較し、定量的な特性比較を行うことができた。離散コサイン変換（DCT）を用いる方式においては、１次元よりも２次元の耐性が高く、更に２次元よりも３次元が高い耐性を持つことが示された。また、Chirp 変換を用いる方式は DCT と同類の直交変換であることから、耐性に関する性能はほぼ同程度であった。しかし、Chirp 変換は非対称で有るため、平坦な画像部分にも有効な埋込みがなされ、DCT では埋め込んでも埋め込まなくても検出ができ識別性が劣るのに比べ、平坦部にも埋込むことができることによる埋込み範囲の拡大という特徴を持つ。Chirp 変換を用いる方式においては、１次元、２次元、３次元の比較を行ったが、特に３次元共に Chirp 変換を用いる 3D-chirp 変換方式は埋込み特性が悪いことが分かった。これは、提案した Chirp 変換が１次元のモデルで設計したためと考えられた。これに関し、多数の検証実験を繰り返した結果、DCT と Chirp 変換を組み合わせる方式は特性がよくなることが分かった。最終的に、２D-DCT と 1D-Chirp 変換を組み合わせた合計で３次元の変換を用いると耐性が最も高く、埋込み範囲も最も広くなるという貴重な結果が得られた。

埋込み誤差については、誤差の逆数の対数評価である信号（S)対誤差ノイズ（N)の比である SN 比の値で評価と比較を行った。埋込みビット数により、誤差が異なるので、１ビットあたりの埋込みに対する誤差に換算して、次元の違いで起こる不整合を調整した。

第４章　統合実験とその他の実験

2 章.3 章で方式の提案と伴に、基本的な実験結果を示してきた。この章では、2、3 章両方にまたがる統合実験やそれ以外の追加実験や動作検証などの実験について述べる。

4－1 実験（1） 表関数を用いた電子透かし埋込方式

難読化の一手段として演算過程を数式やコンピューターのプログラムではなく、表関数を用いて行う方式を提案し、検討した。演算式や処理の流れのプログラムは、最終的には一連のコンピュータープログラムソースコードになり、コンパイラーにより実行プログラムに変換される。ソースコードや実行（オブジェクトコード）プログラム難読化することが行なわれているが、逆アセンブラーにより、オブジェクトコードはソースコードに戻すことも可能である。ソースコードは解析手法により一定の意味のわかるソースコードに変換できる場合もあり、やがて解析される可能性が高い。そこで、この解析を不可能にする手段として表関数を提案してきた。この方式については、2-10 で説明した。ここでは、電子透かしの埋込みと検出ついて行った実験について述べる。

図 4－1 の画像（再掲）の(299、199)から横に４画素を抽出した 4*1 の１次元ブロックの要素４個を変換して、2 個目に埋込む。横に並ぶ４画素について、変換を行う場合を示す。表 4-1 に埋込み処理結果と検出処理の一部を示す。埋込みは 4 カ所しか無いため、交流成分の絶対値平均最大の第二位置に行う。他に DC に入れる等も可能である。変換後の第二位置に変更を施すもので、量子化(Q)=16 にして埋込みを示している。埋込み後のデータに対し、JPEG 圧縮を行って劣化した結果を c、d に示す。検出は、再度変換のみ行い、第二成分の値が、Q は±８の範囲にあれば、検出されたと判定する。簡易検出である。基本特性を調べた。

図 4-1 実験画像（1013kb 、720×480 画素、モノクロ）.

表 4-1　埋込みと検出データ(a-d は実験番号で、表中の数値は逆変後の画素値).

	配列位置	1	2	3	4
a	原画像	48	43	43	45
b	埋込み後	57	47	39	36
c	JPEG 1/20 圧縮	51	44	34	44
d	JPEG 1/25 圧縮	46	46	46	32

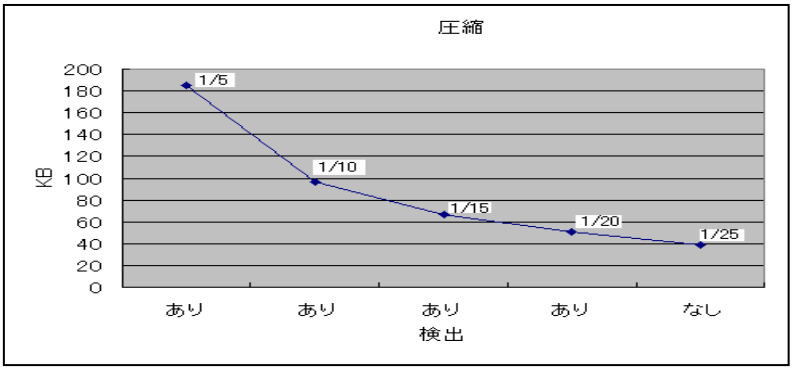


図 4-2(a) 透かしの耐性評価 (4 次 DCT、JPEG 圧縮).

実験環境：耐性の実験はソフト PaintShopPro で JPEG 圧縮することで行う。埋込みはソフトウェア Microsoft Visual C++ 2008 Express Edition を使用した。図 4-2(a)は JPEG による電子透かしの耐性である。横軸は検出の判定の結果であり、縦軸は画像の容量(kb)である。圧縮率は 1/18 まで透かし情報が残っている。

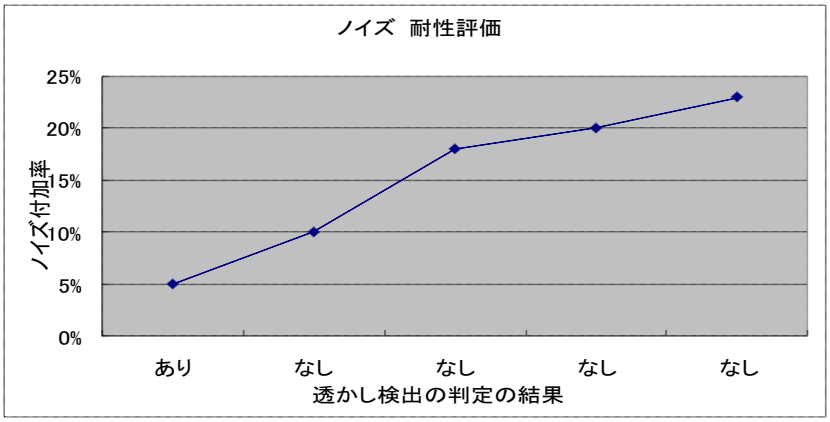


図 4-2(b)　透かしの耐性評価 (4 次 DCT、ノイズ付加).

図４-2(b)はノイズ付加の透かしの耐性である。横軸は透かし検出の判定の結果であり、縦軸はノイズ付加率である。画像に全体的にノイズを加える。ノイズ付加率は 19%まで透かし情報が残っている。静止画像に透かしを埋め込手法により、16 の倍数での変更すれば、数値の変化もそれほど大きくないので、ほぼ原画像と見た所ではにわからない。変更によって、値が大きく変わるのと、変更の単位が大きくなればなるほど、透かし情報を埋め込む際の値の変化も大きくなり、画像の変化も大きくなった。画像（透かしあり）を圧縮(JPG)しても、ノイズを付けても、透かしが残しているから、表関数を用いて埋め込む変更の手法で透かしの耐性があることがわかった。

・考察.結論.今後

４次の変換により、埋込み位置を画像の内側に設定できるため、画像の周辺部を切り取る様な攻撃にも耐性があり、小サイズではあるが、基本性能が確認された。

埋込み位置では、中域の成分が現れる様なものに変更することにより、より効率が上がり、耐性も向上すると考えられる。検出器は 128MB で有るため、さらに埋込み画素のブロックサイズ４を拡大することが可能である。難読化の種類の数は、形式的には、入力のアドレス数である 24 ビットになっている。もともと 4 画素で 32 ビットであるが２画素＋1 画素＋1 画素に分解することにより表の入力は MAX24 ビットに縮退できる。

演算処理の種類は、未知のパラメータを多数含む非線形の関数とすれば、数値計算による推定の解析演算量は膨大になる。今後は、この次数を増加させる方が良い。電子透かしの埋込みと検出のソフトウェアの難読化では、埋込み方式である関数の形式乃至は埋込み処理の形式まで判明しないと、埋込みを除去することができない。

4-2 実験（２）

実験環境:ソフト PaintShopPro で JPEG 圧縮する。ソフトウェア Microsoft Visual C++ 2008 Express Edition。

変換後の第二位置に変更を施すもので、Q はそれぞれ Q=32 にして埋込みを示している。埋込み後のデータに対し、JPEG 圧縮を行って再度変換のみ行い、第二成分の値が、±16 の範囲にあれば、検出されたと判定する。この方式は、参考文献[3-19]にある量子化を用いない方式で、今では再現性が悪いので、使用されることはないが、比較のため実験をした。

図４-3(a)(b)に埋込み位置を示す図 4-4 はノイズを付加した耐性実験の場合で、埋込み後にノイズを加えている。

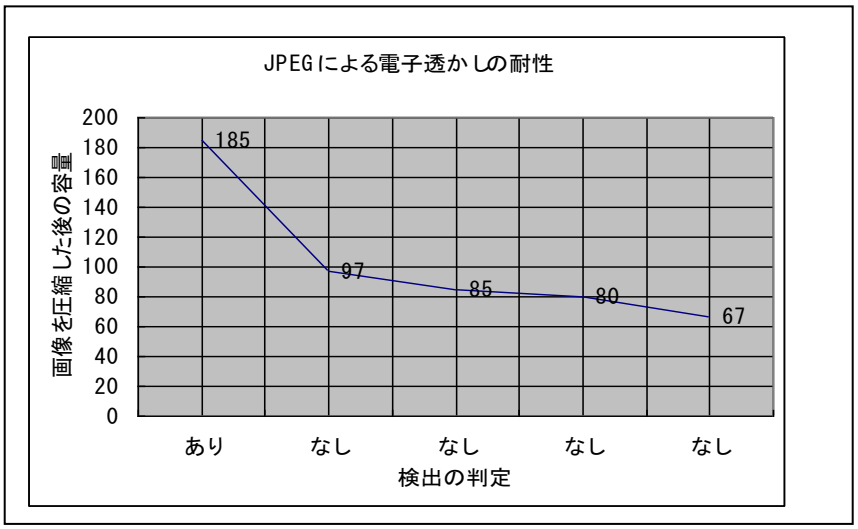
図 4-3(a)　原画像.

図 4-3(b)　埋め込む位置(4 次 DCT、JPEG 圧縮).

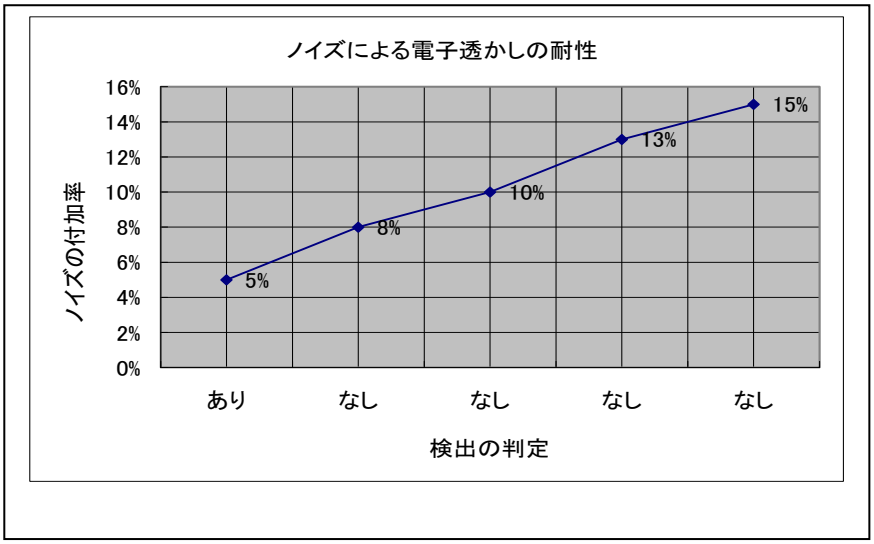
図 4-4　埋め込む位置(4 次 DCT、ノイズ付加).

図 4-5(a)は圧縮後の画像で電子透かしの耐性を調べる目的で圧縮したもので、横軸は検出の判定結果であり、縦軸は画像の容量(kb)である。圧縮率は 1/5 まで透かし情報が残っている。

図 4-5(b)はノイズ付加の透かしの耐性である。横軸は透かし検出の判定の結果であり、縦軸はノイズ付加率である。画像に全体的にノイズを加える。ノイズ付加率は 5%まで透かし情報が残っている。 図 4-6 は電子透かしを埋め込んだ位置の 4 次のブロックの拡大図である。



4-5(a)　JPEG による電子透かしの耐性評価.



4-5(b)　ノイズによる電子透かしの耐性評価.

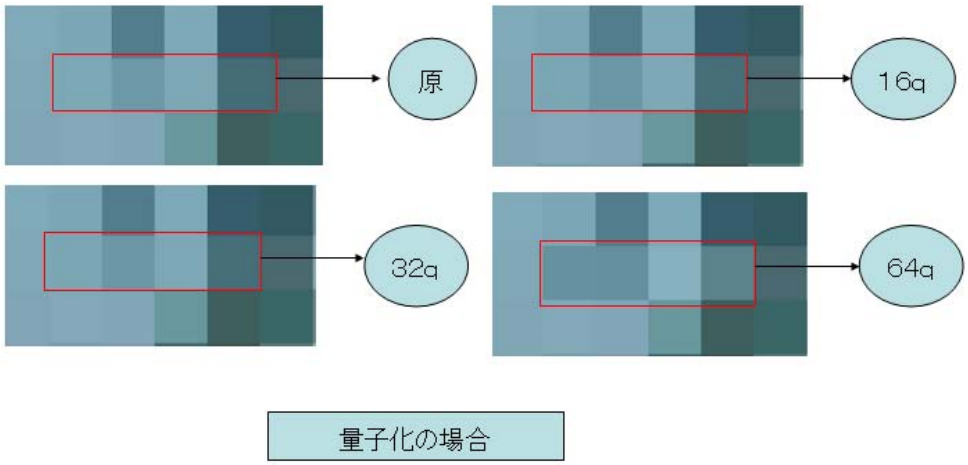


図 4-6 埋め込んだブロックの拡大図.

4－3 実験（3）

同じ実験環境で別の画像で実験を行った。

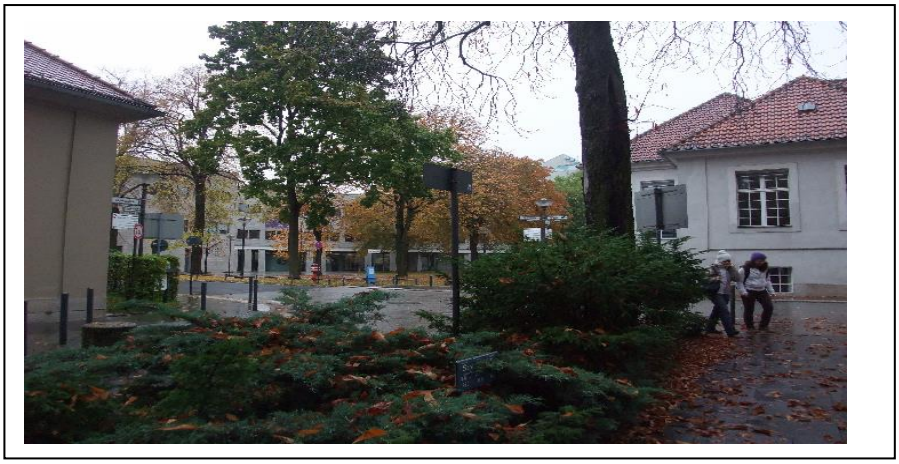
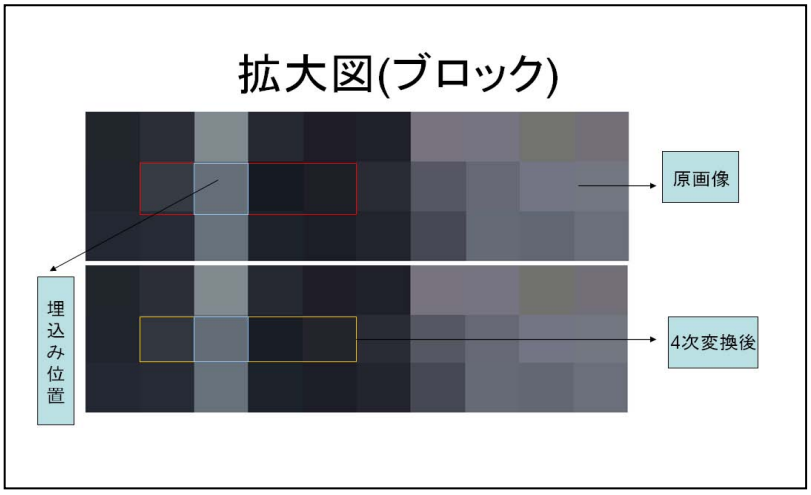


図 4-7　原画像（1013kb 、720×480）.

表 4-2　埋込みと検出データ(a-d は実験番号で、表中の数値は逆変後の画素値).

	配列位置	1	2	3	4
a	原画像	55 59 68	102 110 121	22 26 33	31 31 39
b	埋込み後	50 54 63	100 108 119	25 29 38	36 36 44
c	JPEG 1/12 圧縮	38 43 47	111 121 123	33 42 47	42 49 57
d	JPEG 1/15 圧縮	34 48 57	102 104 119	41 43 58	27 29 44

図 4-8 は電子透かしを埋め込んだ位置の 4 次のブロックの拡大図である
図 4-9(a)は JPEG による電子透かしの耐性である。



4-8　埋め込み位置における画素値の例.

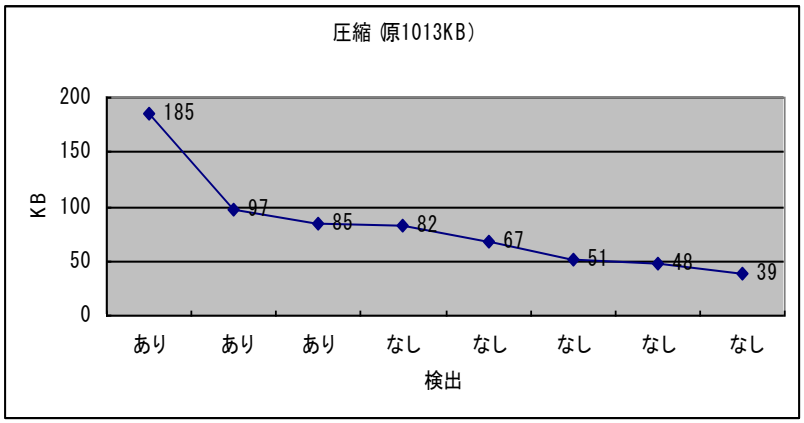


図 4-9(a) 透かしの耐性評価 (4 次 DCT、JPEG 圧縮).

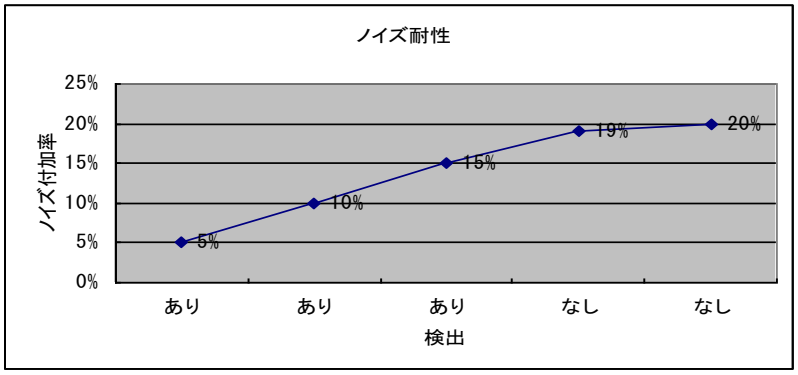


図 4-9(b)　ノイズによる電子透かしの耐性評価(4 次 DCT、JPEG 圧縮).

横軸は検出の判定であり、縦軸は画像の容量(kb)である。圧縮率は１/12 まで透かし情報が残っている。図 4-9(b)はノイズ付加した時の透かしの耐性である。横軸は検出の判定であり、縦軸はノイズ付加率である。画像に全体的にノイズを付加し、ノイズ付加率は 15%まで透かし情報が残っている。静止画像に透かしを埋め込手法により、16 の倍数での量子化すれば、数値の変化もそれほど大きくないので、ほぼ原画像と見た所ではにわからない。量子化によって、値が大きく変わるのと、量子化の単位が大きくなればなるほど、透かし情報を埋め込む際の値の変化も大きくなり、画像の変化も大きくなった。画像（透かしあり）を圧縮(JPG)しても、ノイズを付けても、透かしが残しているから一定の透かしの耐性がある。

4－4 実験（４）

同じ実験環境で別の画像で実験を行った。９枚画像を使用して行った。これは種類の異なる画像に対しても同様な動作がなされていることを確認するものである。

図 4-10 の 9 枚画像の各(300、200)から横に８画素を抽出した 8*1 の 1 次元ブロックの要素８個を変換して、2 個目に埋込む。量子化の幅は Q=32 にして埋込みを示している。埋込み後のデータに対し、JPEG 圧縮を行って再度変換のみ行い、第二成分の値が、 32n±8 の範囲にあれば、検出されたと判定する。

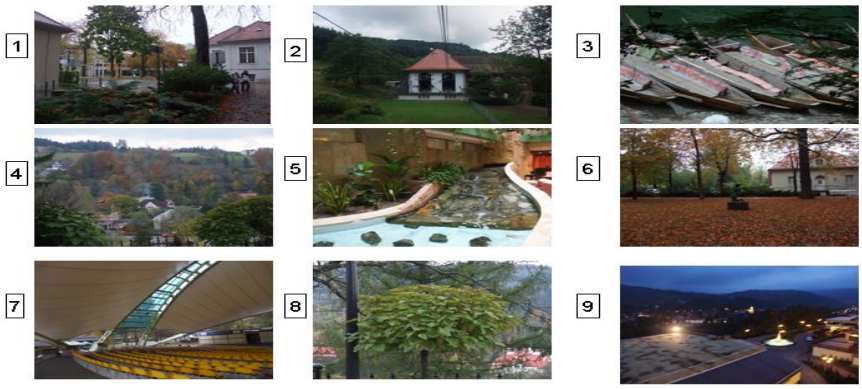


図 4-10　 実験画像（1013kb 、720×480）.

図 4-11 は電子透かしを埋め込んだ位置の８次のブロックの画像８と 5000 倍拡大した図である。各 2 個目の画素を輝度で以下のように示した。

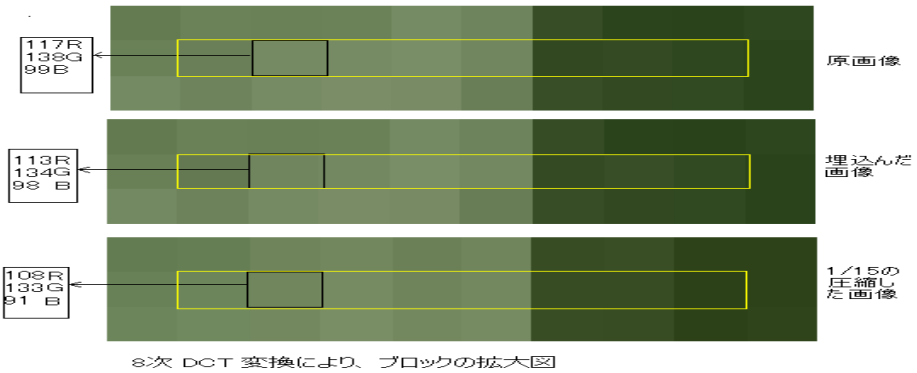


図 4-11 画素値の比較.

図 4-12 は JPEG 圧縮に対する電子透かしの耐性である。横軸は検出の判定であり、縦軸は画像の圧縮率である。画像の模様により、透かし情報が残っている圧縮率の最小値は 10、最大値は 26 であった。画像に透かしを埋め込む手法により、32 の倍数での量子化すれば、

数値の変化もそれほど大きくないので、ほぼ原画像と見た所ではわからない。量子化によって、量子化の単位が大きくなればなるほど、透かし情報を埋め込む際の値の変化も大きくなり、画像の変化も大きくなった。画像（透かしあり）を圧縮(JPG)し、透かしが残っているから、 8 次 DCT による表関数を用いた量子化の手法で透かしの耐性がある。各種画像で試みたが異常はなく、他のテストデータでプログラムの動作も問題ないことも、合わせ一段と確実になった。

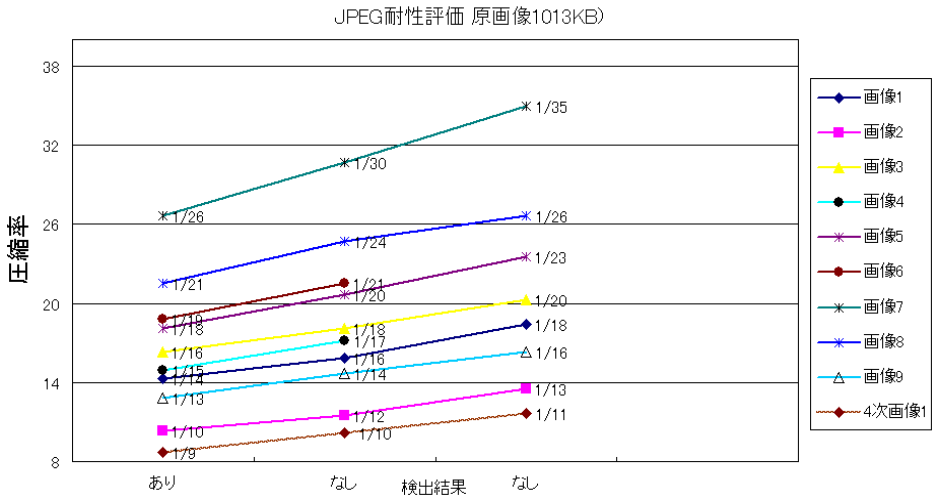


図 4-12 圧縮率の限界付近での検出特性.

4-4 の考察. 結論. 今後

9 枚の画像で実験して、1 次元の埋め込みとしては、1/12 以上の耐性が確保され、画像による変動はあるものの、最低保証が 1/12 あたりであることが示されている。埋込み位置では、中域の成分が現れる様なものに変更することにより、より効率が上がり、耐性も向上すると考えられる。

4－5 実験（５）２次元と１次元の比較

埋込みブロックは、画像の左上から右と下に 200 番目の画素を起点として、横とたてに、各８画素抽出したものとする。これを変換して、(1、0)と(0、1)に２個の量子化により埋め込む。埋込み後のデータに対し、圧縮立の異なる JPEG 圧縮を行って、検出の判定をする

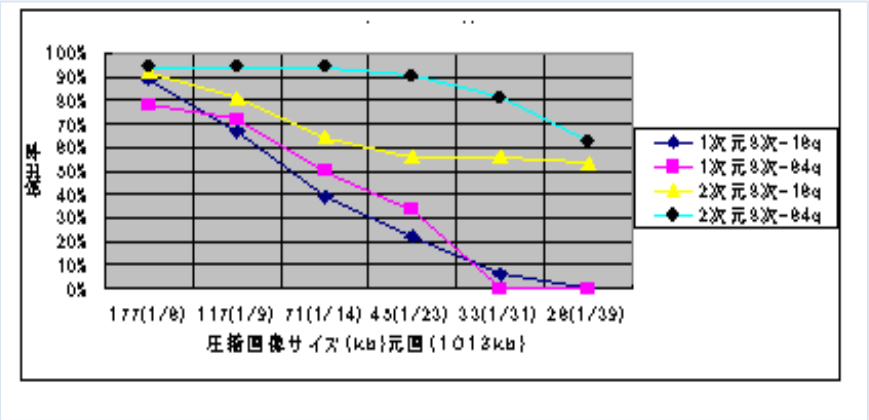


図 4-13(a)　実験画像１の JPEG 圧縮耐性評価.

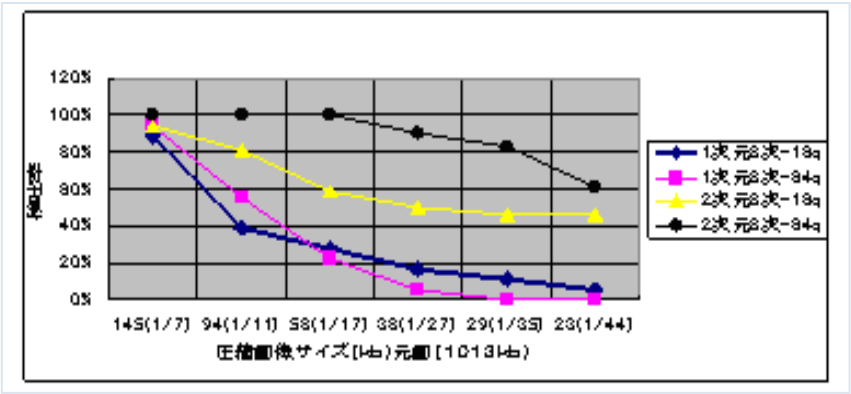


図 4-13(b)　　実験画像３の JPEG 圧縮耐性評価.

図 4-13 の(a)と(b)は JPEG 圧縮に対する電子透かしの耐性である。横軸は圧縮した画像サイズ(kb)であり、縦軸は検出率である。画像の模様により異なっているが、１次元８次 DCT 変換で量子化倍数 16 の場合、透かし情報が残っている。

同様に量子化倍数 64 の場合と、２次元８次 DCT 変換で量子化倍数 16 の場合、透かし情報が残っている。

表 4-3 から、2 次元の方が、1 ビット当たりの性能が向上している事がわかる。この表で、１次元は 18 ビットであるのに対し２次元は３ヵ所で 54 ビットとなるため、１枚の劣化を公平に比較するため、１ビット当たりの劣化に換算した値も示してある。係数が正しく調整されていないため、ここでは２次元の方がやや劣化大きくなっている。

１次元では画像中に 18 か所のブロックをとり、18 ビットの埋め込みを行った。２次元では８×８ブロックで、３か所（３ビット）埋め込み、18x3=54 ビット埋め込んだ。

表 4-3　埋込みビット数と S/N（画像３、１枚）.

Q=64	S/N	ビット数	１ビット当たり
１次変換	77.6dB	18	90.1dB
２次変換	63.2dB	54	82.6dB

4-5　まとめ

DCT を用いた電子透かしの埋込み器と検出器を構成した。8 次の変換により、埋込み位置を画像の内側に設定できるため、切り取りの攻撃に耐性がある。1 次元より性能が上がった。今後、３次元変換にする。表 4-3 のビット配分のように次元が違うと画像の中の埋め込み位置と埋め込み可能な周波数面での係数の位置に制約が出るため、同一ビットに揃えることはできない。ここでは、換算しながら１次元、２次元の比較を行った。

4－6 実験（６）Chirp 変換の場合

電子透かしの埋込みは、線形変換と量子化によって行う。１次元変換の場合は静止画から切り出した 1×８画素のブロックに対し、Chirp 変換する。データは低域に集中するので、DC の次の第 2 番目に埋込み処理としての量子化を行う、透かし情報は 1 ビットとなる。3 次元の場合は、３次元の３つの要素を(x、y、z)とした時、(0、0、0)と最も DC から遠い(1、1、1)を除いた 6 か所への埋込みを行う。これにより 6 ビットの埋込みを行うことができる。埋込み情報量と誤差(S/N)評価の比較を行う。

埋込み後のデータに対し、圧縮率の異なる JPEG 圧縮を行って、検出の判定をする。

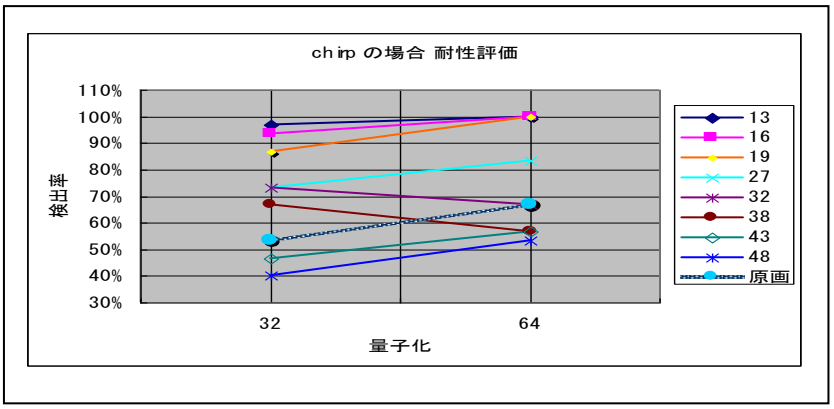


図 4-14　JPEG 圧縮耐性評価.

図 4-14 は JPEG 圧縮に対する電子透かしの耐性である。横軸は量子化の倍数であり、縦軸は検出率である。右の各線の数字は圧縮の比率（何分の１を表す）である。画像の模様により、3 次元８画素 Chirp 変換で、量子化倍数 32 と 64 の場合、圧縮は 1/27 の時は 70%

透かし情報が残っている。表 4-4 は埋込みビット数と S/N の比較である。

表 4-4　埋込みビット数と S/N Chirp の場合.

	S/N	ビット数	１ビット当たり
Q=32	83.4　dB	6	91.2dB
Q=64	81.4　dB	6	89.2dB

4-6 のまとめ

Chirp を用いた電子透かしの埋込み器と検出器を構成した。S/N と耐性は DCT とほぼ同一で、同じ線形変換として大きい差はなかった。Chirp 変換は画素平らの部分に電子透かしを埋込むことができる特徴を有する。一方ＤＣＴは平坦部では埋め込みをしようとする係数が０になって、埋め込みができないという不都合がある。Chirp 変換埋め込み場所を画像の状態によらず自由に設定できる、という特徴を持っている。

4－7 実験(7)　３次元 DCT の場合

実験用画像

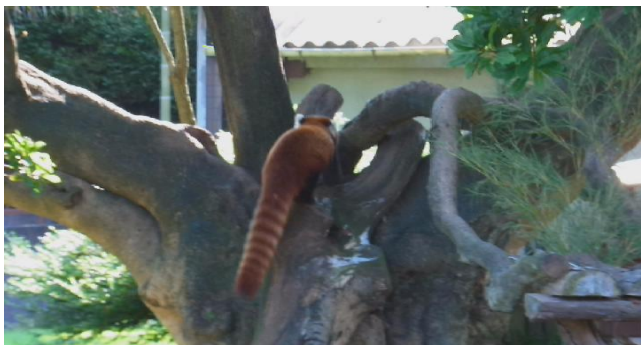


図 4-15　原画像(1013kb)、ppm.

埋込みブロックは、画像の左上から右と下に 199 番目の画素を起点として、横に８画素抽出したものとする。これを変換して、2 個目に量子化により埋め込む。埋込み後のデータに対し、圧縮率の異なる JPEG 圧縮を行って、検出の判定をする。

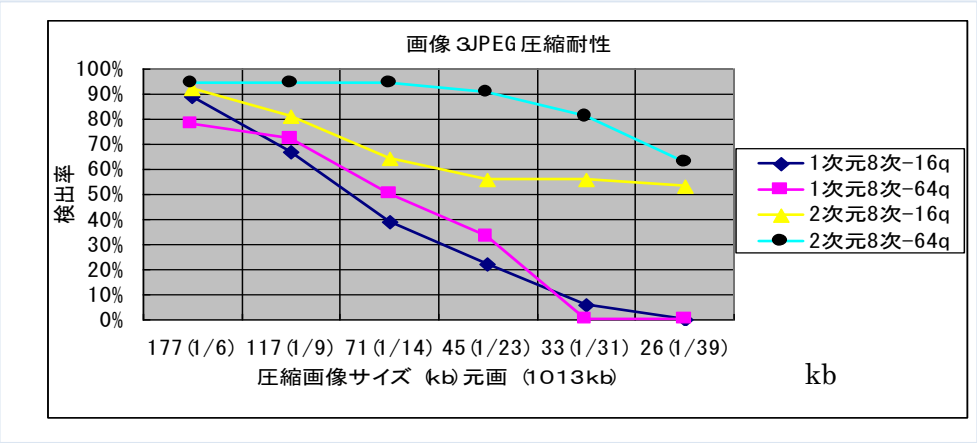


図 4-16　JPEG 圧縮耐性評価.

図 4-16 は JPEG 圧縮に対する電子透かしの耐性である。横軸は圧縮した画像サイズ(kb)であり、縦軸は検出率である。画像の模様により、１次元８次 DCT 変換で量子化倍数 16 の場合、透かし情報が残っている。

同様に量子化倍数 64 の場合、透かし情報が残っている２次元８次 DCT 変換で量子化倍数 16 の場合、透かし情報が残っている。表 4-4 から、2 次元の方が、1 ビット当たりの S/N 性能が向上している事がわかる。

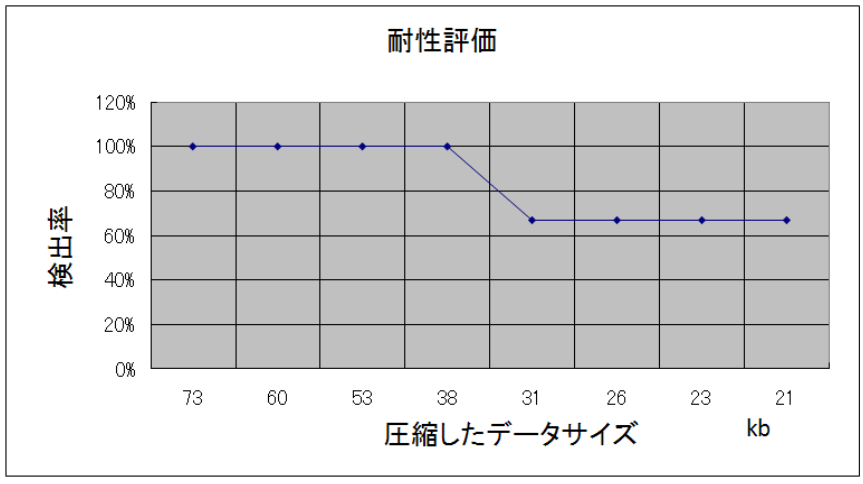


図 4-17　3 次元の JPEG 圧縮耐性比較

図 4-17 は 3 次元 DCT 変換による電子透かしの JPEG 圧縮耐性評価である。縦軸は検出率であり、横軸はで圧縮したデータあり、グラフの中の数字は圧縮率である。

4-7 のまとめ

DCT を用いた電子透かしの埋込み器と検出器を構成した。3 次元 8 画素の変換により、埋込み位置を画像の内側に設定できるため、切り取りの攻撃に耐性がある。3 次元は 2 次元より性能が上がった。又、量子化に耐性のある方式を得ることができた。これらの結果は本方式の量子化パラメータ設計において役立てることできた。

第５章　まとめ、今後の課題

２章では、途中結果を復号鍵とする暗号化の方式は、暗号化理論により、確実に計算量が増大する難読化方式である。途中結果を復号鍵とする暗号鍵を求めることができる。十進数 8 桁の計算時間の実測は約 0.5 秒で、計算時間が見積もれるようになった。計算量を見積もる難読化方式に加え、計算アルゴリズムを隠蔽する ROM 方式の難読化方式を検討し、実際に実現構成を行い、モノクロ画像での電子透かし埋込み実験を行った。埋込み領域が小さいため、周囲の切取りにも耐性があることが特徴である。一方、埋込みを拡大するには、ROM 容量の限界が有り、難読性の強さを減らして、階層的な形式にしていくことが考えられる。

３章では、変換に DCT を用いて、１、２、３次元変換後に量子化による埋込みを行う電子透かし方式を比較した。従来単独の結果しか得られていなかったが、埋込みビット数を揃え比較し、定量的な特性比較を行うことができた。離散コサイン変換（DCT）を用いる方式においては、１次元よりも２次元の耐性が高く、更に２次元よりも３次元が高い耐性を持つことが示された。また、Chirp 変換を用いる方式は DCT と同類の直交変換であることから、耐性に関する性能はほぼ同程度であった。しかし、Chirp 変換は非対称で有るため、平坦な画像部分にも有効な埋込みがなされ、DCT では埋め込んでも埋め込まなくても検出ができ識別性が劣るのに比べ、平坦部にも埋込むことができることによる埋込み範囲の拡大という特徴を持つ。Chirp 変換を用いる方式においては、１次元、２次元、３次元の比較を行ったが、特に３次元共に Chirp 変換を用いる 3D-chirp 変換方式は埋込み特性が悪いことが分かった。これは、提案した Chirp 変換が１次元のモデルで設計したためと考えられた。これに関し、多数の検証実験を繰り返した結果、DCT と Chirp 変換を組み合わせる方式が特性がよくなることが分かった。最終的に、２D-DCT と 1D-Chirp 変換を組み合わせた合計で３次元の変換を用いると耐性が最も高く、埋込み範囲も最も広くなるという貴重な結果が得られた。

埋込み誤差については、誤差の逆数の対数評価である信号（S)対誤差ノイズ（N)の比である SN 比の値で評価と比較を行った。埋込みビット数により、誤差が異なるので、１ビットあたりの埋込みに対する誤差に換算して、次元の違いで起こる不整合を調整した。また、変換に用いた係数精度の詳細な検討などを行った。

４章では、2、3 章にまたがる統合方式の実験、その他文献に記載された他の方式の追試実験、埋込みによる誤差の評価、他の多数の画像での実験を行い有効性を確認した。

今後は、難読化については、サイズの小さいブロックで ROM 方式によるアルゴリズムの隠蔽を行う手法を開発してきたが、サイズの大きい領域への展開を行っていくことも重要で

ある。また、特殊関数として、電子透かしに適合したものを検討することも有効である。３D 電子透かしについては、２D-DCT と１D-Chirp 変換を組み合わせた方式が最適であったが、今後は新たな変換や、Chirp 変換に対しては、2、3 次元にも適用可能なモデルの構築と、変換係数の開発など重要なことがらである。

論文リスト

学術雑誌（査読あり）

[1]Yuanyu Wei, Kazuo Ohzeki.”Two Obfuscation Methods by Controlling Calculation Amounts and by Table Function for Watermarks”. IJCSA 01/2011; 8:110-122

国際会議プロシーディング（査読あり）

[2]Yuanyu Wei and Kazuo Ohzeki, “Obfuscation Methods with Controlled Calculation Amounts and Table Function”, Proc. 3rd International Symposium on Multimedia ? Applications and Processing (MMAP), pp.775-780, Oct. 2010.

[3]Yuanyu Wei and Kazuo Ohzeki, “A New Obfuscation Method Using Random Functions”, Proceedings of Telecommunications, Networks and Systems (TNS), IADIS of MCCSIS July, 2010

学会

[4]魏遠玉、大関和夫、「表関数による難読化方式を用いた電子透かし埋込方式における量子化特性の考察」P1-1, 2011 年度 画像電子学会第 39 回年次大会　2011 年　6 月 25 日

[5]魏遠玉、大関和夫、「DCT による難読化を用いた電子透かしの応用例の考察」、第 17 回創発システム・シンポジウム「創発夏の学校 2011」予稿 pp. 59-61　2011 年 8 月、計測自動制御学会システム・情報部門

[6]魏遠玉、大関和夫、平川豊、「難読化を用いた 8 次 DCT による電子透かし埋込み方式」2011 年度映像メディ処理シンポジウム、pp.109-110、IMPS-I5-12、2011 年 10 月 26-28 日

[7]魏遠玉、大関和夫、「表関数による難読化方式を用いた電子透かし埋込方式」、情報処理学会全国大会、セキュリティ部門、6Y-3、2011 年 3 月

[8]魏遠玉、大関和夫 情報処理学会第 73 回全国大会　6y-3-(561-562)
表関数による難読化方式を用いた電子透かし埋込方式　3/10 2012

[9]魏遠玉、大関和夫、平川豊、佐藤清次、「電子透かし埋込む方式に於ける 1 次元,2 次元,3 次元の DCT 性能比較」2013 年度画像電子学会第 41 回年次大会 P-7　６月２３日

研　究　業　績　書

学術雑誌（査読あり）

[1]Kazuo Ohzeki, Yuki Seo, Engyoku Gi: Discontinuity in SVD Embedding Mapping Used for Watermarks. IJCSA 7(3): 9-17 (2010/7)

[2]YuanYu Wei,and　Kazuo Ohzeki "TWO OBFUSCATION METHODS BY CONTROLLING CALCULATION AMOUNTS AND BY TABLE FUNCTION FOR WATERMARK" International Journal of Computer Science and　pplications,Technomathematics Research Foundation Vol. 8, No. 1, pp. 110-122, 2011/7.

[3]Kazuo Ohzeki ,Yuan yu Wei,Hiroki Okumura,Ulrich Speidel[A LOSSLESS RE-ENCODING OF MPEG-2 CODED FILE BY　INTEGRATING FOUR MOTION VECTORS] JCSA 8(1):14-35. 2011/07.

国際会議プロシーディング（査読あり）

[4]Yuanyu Wei and Kazuo Ohzeki, “A New Obfuscation Method Using Random Functions”, Proceedings of Telecommunications, Networks and Systems (TNS), IADIS of MCCSIS July, 2010.

[5]Kazuo Ohzek, Engyoku Gi,Ulrich Speidel, "LOSSLESS RE-ENCODING OF MPEG CODED FILES USING V-V CODES", Proceedings of CG and Visalization, Computer Vision and Image Processing, IADIS of MCCSIS July, 2010.

[6]Yuanyu Wei and Kazuo Ohzeki, “Obfuscation Methods with Controlled Calculation Amounts and Table Function”, Proc. 3rd International Symposium on Multimedia ? Applications and Processing (MMAP), pp.775-780, Oct. 2010.

[7]Kazuo Ohzeki, YuanYu Wei, Eizaburo Iwata and Ulrich Speidel, “Basic Consideration of MPEG-2 Coded File Entropy and Lossless Re-encoding”, Proc. 3rd International Symposium on Multimedia　Applications and Processing (MMAP), pp.741-748, Oct. 2010

[8]Kazuo Ohzeki, Hiroki Okumura, Yuan Yu Wei, Eizaburo Iwata, Ulrich Speidel, “A LOSSLESS RE-ENCODING OF MPEG-2 CODED FILE BY INTEGRATING FOUR MOTION VECTORS”, International Journal of Computer Science and Applications 2011: Volume VIII Issue I, (Special

Issue on Multimedia　Applications), Technomathematics Research Foundation pp.14-35, Aug. 2011.

[9]Kazuo Ohzeki, YuanYu Wei, Yutaka Hirakawa and Kiyotsugu Sato, “A New Watermarking Method with Obfuscated Quasi-Chirp Transform”, 10th International Workshop on Digital-forensics and Watermarking (IWDW11), Session: Watermarking 2, Atlantic City NJ. Oct. 22-26, 2011

[10]Kazuo Ohzeki,Yuan yu Wei ,Yutaka hiraakawa,KIyotsugu Sato,[Consideration of the Watermark Inversion Attack and its Invalidation Framework]IWDW 2012 (11th International Workshop on Digital-forensics and Watermarking)2012/11

[11]Kazuo Ohzeki,Yuan yu Wei ,Yutaka hiraakawa,KIyotsugu Sato,[TWO-STAGE WATERMARK SYSTEM FOR INCREASING COPYRIGHT POTENCY IN IMAGE MEDIA]IADIS e-Commerce,Proc. of Iadis e-Commerce (USB),2013/6

[12]Ohzeki, Kazuo, Wei, Yuan Yu, Hirakawa, Yutaka, Sato, Kiyotsugu. (2013, Aug).Consideration of the Watermark Inversion Attack and its Invalidation Framework.Proc. of International Workshop on Digital-Forensics and Watermarking IWDW2012 doi 10.1007/978-3-642-40099-5-9.

[13]Kazuo Ohzeki1, Kazutaka Bannai, Yuan-Yu Wei1, Yutaka Hirakawa1 and iyotsugu Sato [Robust Watermarking Method By Systematic Block Diffusion Using Discrete Cosine Transform] 14th Australian Information Warfare Conference.2013.12

国内発表

[14]大関 和夫、加藤 剛、魏 遠玉、「ＶＶ符号による MPEG 圧縮ファイルのロスレス再符号化方式の基本検討」電子情報通信学会　画像工学研究会　技術報告　IE2010-6, pp.31-36, April, 2010.

[15]大関和夫、魏　遠玉、「表関数を用いた電子透かしの検出器ソフトウェアの難読化」情報処理学会、第 51 回コンピュータセキュリティ研究発表会、 Vol.2010-CSEC51　No.1 pp.1-6,　Dec.10.2010.

[16]魏遠玉、大関和夫、「表関数による難読化方式を用いた電子透かし埋込方式」、情報処理学会全国大会、セキュリティ部門、6Y-3、2011 年 3 月.

[17]大関和夫 、魏遠玉、「疑似 Chirp 変換による埋込みと難読化を用いた電子透かし」情報処理 学会、コンピュータセキュリティ研究会(CSEC),Vol.2011-CSEC-52 No.29

pp.1-10.,2011 年 3 月
[18]魏遠玉、大関和夫、「DCT による難読化を用いた電子透かしの応用例の考察」、第 17 回創発システム・シンポジウム「創発夏の学校 2011」予稿 pp. 59-61　2011 年 8 月、計測自動制御学会システム・情報部門
[19]魏遠玉、大関和夫、「表関数による難読化方式を用いた電子透かし埋込方式における量子化特性の考察」P1-1, 2011 年度 画像電子学会第 39 回年次大会　2011 年　6 月 25 日
[20]魏遠玉、大関和夫、平川豊、「難読化を用いた 8 次 DCT による電子透かし埋込み方式」2011 年度映像メディ処理シンポジウム、pp.109-110、 IMPS-I5-12、2011 年 10 月 26-28 日
[21]大関和夫、魏　遠玉、平川　豊・佐藤清次、[電子透かしに対するインバージョンアタックの考察と丸め演算による対策]電子情報通信学会 研究会,SIS, IPSJ-AVM 2012/08
[22]大関和夫、魏　遠玉、倉木真生、平川　豊、佐藤清次[ブロック拡散と Biased-Chirp 変換を用いた電子透かし]電子情報通信学会 セキュリティ研究会　2012/03
[23]魏遠玉、大関和夫、平川豊、佐藤清次、「チャープ変換を用いた動画用 3 次元電子透かし」情報通信学会、学生大会、2012 年 6 月
[24]魏遠玉、大関和夫、平川豊、佐藤清次「電子透かし埋込む方式に於ける 1 次元,2 次元,3 次元の DCT 性能比較」画像電子学会年次大会 2013 年 6 月
[25]大関和夫、坂内一貴、魏　遠玉、平川　豊、佐藤清次[組織的ブロック拡散と DCT を用いた高耐性電子透かし]電子情報通信学会 研究会,SIS, IPSJ-AVM 2013/9

参考文献

[2-1] Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S. and Yang, K.: On the (Im)possibility of Obfuscating Programs, pp. 1-18, CRYPTO (2001).

[2-2] S. Chow, et al., “White-Box Cryptography and an AES Implementation”, 9th WS SAC pp.250-270, Springer LNCS 2595, 2002.

[2-3]大関和夫、叢力、「計算量的難読化を仮定した、不特定第三者の認証に依存する電子透かし方式」情報処理学会，研究報告，CSEC-32， pp. 61-66, 2006 年 3 月.

[2-4]門田 暁人 、 高田 義広 、 鳥居 宏次「ループを含むプログラムを難読化する方法の提案」電子情報通信学会論文誌．D-I，情報・システム，I-コンピュータ J80-D-1(7), 644-652, 1997-07-25

[2-5] Christian Collberg , Clark Thomborson , Douglas Low,” A Taxonomy of Obfuscating Transformations (1997) Cached” Technical Report TR148.pdfDepartment of Computer Science The University of Auckland 1997.

[2-6] Yuanyu Wei and Kazuo Ohzeki,“A New Obfuscation Method Using Random Functions ” proceedings of Telecommumications,Networks and Systems, IADIS International conferences. P263-265,July 2010.

[2-7] Yuanyu Wei and Kazuo Ohzeki, “Obfuscation Methods with Controlled Calculation Amounts and Table Function”, Proc. 3rd International Symposium on Multimedia ? Applications and Processing (MMAP), pp. 775 -780 Oct. 2010.

[2-8] YuanYu Wei,and Kazuo Ohzeki "TWO OBFUSCATION METHODS BY CONTROLLING CALCULATION AMOUNTS AND BY TABLE FUNCTION FOR WATERMARK" International Journal of Computer Science and Applications, Technomathematics Research Foundation Vol. 8, No. 1, pp. 110 – 122, 2011

[2-9]暗号技術大全 19 章公開アルゴリズム

[2-10] Kiyomoto, S., and Tanaka, T.,: Evaluation of Mutational Capability and Real-Time Applicability of Obfuscation Techniques, EICE transactions on fundamentals of electronics, communications and computer sciences E89-A(1) pp.222-226 (2006)

[2-11] Fukushima, T. Tabata, and K. Sakurai.: Proposal and Evaluation of Obfuscation Scheme for Java Source Codes by Partial Destruction of Encapsulation, Proc. of International Symposium on

Information Science and Electrical Engineering (ISEE2003), pages 389–392, Fukuoka, Japan, November 14–15.

[2-12]大関和夫、魏　遠玉、「表関数を用いた電子透かしの検出器ソフトウェアの難読化」情報処理学会、第 51 回コンピュータセキュリティ研究発表会、 Vol.2010-CSEC51　No.1 pp.1-6,　Dec.10.2010.

[3-1] G. Louizis, A.Tefas, and I. Pitas, "Copyright protection of 3D images using watermarks of specific spatial structure ", IEEE International Conf. on Multimedia and Expo (ICME), Vol.2 pp.557-560, 2002.

[3-2]Y. Wang, A. Pearmain, "Blind MPEG-2 video watermarking robust against scaling", International Conference on Image Processing, ICIP '04., Vol. 4, pp.2159 - 2162, Oct. 2004.

[3-3] Hyun Park, Sung Hyun Lee, Young Shik Moon, "Adaptive Video Watermarking Utilizing Video Characteristics in 3D-DCT Domain", Digital Watermarking, Lecture Notes in Computer Science Volume 4283, pp 397-406, 2006.

[3-4] P. Campisi, A. Neri, “Video Watermarking using quantization index modulation in the 3D-DCT domain”, Second International Workshop on Video Processing and Quality Metrics (VPQM), Scottsdale, Arizona, USA, January 23-25, 2006.

[3-5] Liyun Wang, Hefei Ling; Fuhao Zou ; Zhengding Lu, "Real-Time Compressed- Domain Video Watermarking Resistance to Geometric Distortions ", IEEE MultiMedia Vol. 19 , Issue: 1, pp.70-79, Jan, 2012.

[3-6] F. Perez-Gonzalez, C. Mosquera, M. Barni, A. Abrardo, " Rational dither modulation: a high-rate data-hiding method invariant to gain attacks", IEEE Trans Signal Processing, Vol. 53 , pp. 3960 - 3975, Oct. 2005.

[3-7] Patrizio Campisi, Alessandro Neri, "3D-DCT Video watermarking using quantization–based methods", European Signal Processing Conference (EUSIPCO) 2007, pp.2544-2548, Sept., 2007.

[3-8] Yong-Gang Fu, "Robust video watermarking scheme based on 3D DCT ", International Conf. on Computer Application and System Modeling (ICCASM), Vol. 11, pp.V11-635-638, Oct. 2010.

[3-9] 大関和夫 、魏遠玉、”疑似 Chirp 変換による埋込みと難読化を用いた電子透かし”　情

報処理学会、コンピュータセキュリティ研究会(CSEC),Vol.2011-CSEC-52 No.29 pp.1-10.,2011 年 3 月

[3-10] 魏遠玉、倉木真生、平川豊、大関和夫、佐藤清次、”3 次元 DCT による電子透かし方式に向けた 1,2 次元の性能比較”,2012 年電子情報通信学会総合大会, D-21-4, 2012 年 3 月.

[3-11] 魏遠玉、平川豊、大関和夫、佐藤清次、 “電子透かし埋込み方式に於ける 1 次元,2 次元,3 次元の DCT 性能比較” 画像電子学会年次大会,7P, 7,月 2013

[3-12] Pratt, W., Chen, W., Welch, L. "Slant Transform Image Coding", IEEE Trans. COM-22, pp.1075-1093, Aug.1974.

[3-13] Ohzeki et al., “Obfuscation of Software for Watermark Detector Using Table Function “, IPSJ SIG Technical Report Vol.2010-CSEC-51 No.1pp.1-6, Dec. 2010.

[3-14] Jan Kodovský, Jessica Fridrich, "Calibration Revisited,"Proc. ACM Multimedia and Security Workshop, Princeton, NJ, September 7–8, pp. 63–74, 2009.

[3-15] 山根延元、森川良孝、成相剛士、鶴原篤、”斜交軸上の DCT による画像の高能率符号化法",電子情報通信学会論文誌. B-I,J81-B-1(2), pp.110-117, 1998 年 2 月

[3-16] Jiri Fridrich, "Combining low-frequency and spread spectrum watermarking", Proc. SPIE Int. Symposium on Optical Science, Engineering, and Instrumentation, pp.2-12.July, 1998.

[3-17] Jiri Fridrich, Lt Arnold C. Baldoza and Richard J. Simard, "Robust Digital Watermarking Based on Key-Dependent Basis Functions", Proc. 2nd Information Hiding Workshop, LNCS vol. 1525, Springer-Verlag, New York,pp. 143-157 1998.

[3-18]Kazuo Ohzeki, YuanYu Wei, Yutaka Hirakawa and Kiyotsugu Sato, “A New Watermarking Method with Obfuscated Quasi-Chirp Transform”, 10th International Workshop on Digital-forensics and Watermarking (IWDW11), Session: Watermarking 2, Atlantic City NJ. Oct. 22-26, 2011

[3-19] Yonggang Fu ,"Robust Image Watermarking Scheme Based on 3D-DCT", Sixth International Conference on Fuzzy Systems and Knowledge Discovery, 2009. FSKD '09. pp.437 - 441 Aug. 2009

謝辞

この研究を遂行するにあたり，終始暖かく見守って下さった指導教員の芝浦工業大学大学院の大関和夫博士に深く感謝いたします。日頃から研究の進み具合を気にかけていただき，助言と激励をくださった優しい言葉で私を励まして下さいました。論文の日本語の修正や、情報、資料提供から始まり、研究についてのアドバイスなどの大変面倒を見て下さいました。また快く調査の実施，データの検証にご協力をいただいたばかりでなく，貴重な時間をさいて私の面倒を見てくださいました。大関和夫教授からは指導教官として、優しくも暖かいご指導をいただきました。本論文は、今から考えると学問的な厳密さを追究するには難しいテーマだったようで、何度も壁に突き当たっては挫けそうになった４年間を大関先生は励ましつつ辛抱強く導いてくださいました。博士論文とそれに関連する提出書類のことについて、相談にのっていただきました。本論文をまとめるにあたり、昼夜にわたり御指導、御討論いただいた芝浦工業大学大学院工学研究科制御システムアルゴリズム研究室の大関和夫教授に深く感謝いたします。本当に有り難うございました。

そして合同ゼミでの学部生，院生の多大な協力と励まして，本研究を進める過程で有益な助言御協力をいただいたたくさんの後輩達に感謝の意を表します。学会においてさまざまな先生方々、研究員などに助言と激励をいただいたことは、本研究に大きく影響しています。ここですべての方のお名前を挙げることはできませんが、その中でも特にお世話になった方は、有益な議論をさせていただきました。

また、貴重なご指導とご助言を頂いた副指導教員の大学理工学部学科の杉本徹教授博士、平川豊教授博士、木村昌臣教授博士、産業技術短期大学の佐藤清次教授博士に心より感謝申しあげます。

今回の論文を完成するにあたり、多くの人々にご協力をいただきました。先生方々や研究室の方々の暖かいご協力に感謝の意を表したいと思います。ご多忙のなか、論文の作成にあたり適切なアドバイスと膨大な資料を快く提供してくださいましたこの短い作成期間にもかかわらず、さまざまなご指示とご協力ならびにご便宜を賜り、応援してくださいました全ての方々に支えられ、おかげさまで無事論文完成に至りました。

また、大関教授には、大学院博士課程 4 年間、研究や技術だけでなく今後社会人として必要な様々な経験、礼儀まで御指導いただき、こころより感謝いたします。

最後に、助けていただいた多くの皆様心から感謝しております。先生方々のこれからのますますのご発展とご健勝をお祈り申しあげながら、心から謝意を述べさせていただきます。

付録

主なプログラム資料

本研究の実験で開発した実験用プログラムのソースコードを以下に載せる。３D 変換による電子透かし埋込み方式のソースで、検出用は別のプログラムだか、構成はほぼ同一で、検出パラメータをを変化させ検出するようにしている。他に、色変換の逆行列等の演算は、Octave におけるコマンド処理で行った。

３ｄ（完全版）.cpp

```
// 3d(完全版).cpp : コンソール アプリケーションのエントリ ポイントを定義します。
//

#include "stdafx.h"

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include <fstream>  // 追加 メモ帳
#include <string>

#define BUFF_MAX 65536 /*データ個数*/
#define OPT        0 /*ＯＰＴ　１　光学的ＤＦＴ（直流分がまん中）*/
                     /*ＯＰＴ　０　通常のＤＦＴ（直流分が左はじ）*/
//#define X_SIZE 256//水平画素数、２のべき乗個
//#define Y_SIZE 256//垂直画素数、２のべき乗個
#define EXH 8
#define EXV 8
#define MAX 255
#define MAX_BRIGHTNESS 255
#define block_size_x 8
#define block_size_y 8
#define high  480
#define width 720  //704
#define kk 8


int pbm_getint(FILE* file);
char pbm_getc(FILE* file);


void save_image_data_head(FILE *f_out);
void yuv_rgb(unsigned char image_r[high][width],unsigned char image_g[high][width],
                            unsigned char image_b[high][width],double y[high][width],
                            double u[high][width],double v[high][width]);
void kaki(FILE *f_out,unsigned char image_r[high][width],
                    unsigned char image_g[high][width],unsigned char image_b[high][width]);
```

```
char pbm_getc(FILE* file)
{
  char ch;

  ch = getc(file);

  if (ch=='#')
  {
    do
    {
      ch = getc(file);
    }
    while (ch!='¥n' && ch!='¥r');
  }

  return ch;
}

int pbm_getint(FILE* file)
{
  char ch;
  int i;

  do
  {
          ch = pbm_getc(file);
  }
  while (ch==' ' || ch=='¥t' || ch=='¥n' || ch=='¥r');

  i = 0;
  do
  {
    i = i*10 + ch-'0';
    ch = pbm_getc(file);
  }
  while (ch>='0' && ch<='9');

  return i;
}

 void read_ppm(FILE* file,unsigned char r[high][width],unsigned char g[high][width],unsigned char
b[high][width])
{
         int i,j;

         for(i=0;i<high;i++){
                  for(j=0;j<width;j++){
                           r[i][j]=getc(file);
                      g[i][j]=getc(file);
                      b[i][j]=getc(file);
                  }
         }
}
```

```
void Y_U_V(unsigned char r[high][width],unsigned char g[high][width],unsigned char
b[high][width],
           double y[high][width],double u[high][width],double v[high][width])
{
         int i,j;

         for(i=0;i<high;i++){
                  for(j=0;j<width;j++){
                  y[i][j] = (double)(0.29891*r[i][j]+0.58661*g[i][j]+0.11448*b[i][j]);
                  u[i][j] = (double)(-0.16874*r[i][j]-0.33126*g[i][j]+0.50000*b[i][j]);
                  v[i][j] = (double)(0.50000*r[i][j]-0.41869*g[i][j]-0.08131*b[i][j]);
                  }
         }
}

 void save_image_data_head(FILE *f_out)
{

         fputs("P6¥n",f_out);

         fputs("# Created by michi&KJ from ohzeki lab¥n",f_out);

         fprintf(f_out,"%d %d¥n",width,high);
         fprintf(f_out,"%d¥n",MAX_BRIGHTNESS);

}


void yuv_rgb(unsigned char image_r[high][width],unsigned char image_g[high][width],
                           unsigned char image_b[high][width],double y[high][width],
                           double u[high][width],double v[high][width])
{

         int i;
         int j;

         for(i=0;i<high;i++){
                  for(j=0;j<width;j++){

                                    image_r[i][j] = (unsigned char)(y[i][j]+1.40200*v[i][j]);
                                    image_g[i][j] = (unsigned
char)(y[i][j]-0.34414*u[i][j]-0.71414*v[i][j]);
                                    image_b[i][j] = (unsigned char)(y[i][j]+1.77200*u[i][j]);
                  }
         }
         printf("2.7¥n");
}


void kaki(FILE *f_out,unsigned char image_r[high][width],
                  unsigned char image_g[high][width],unsigned char image_b[high][width])
{
```

```
        int x,y;
        for(x=0;x<high;x++){
                for(y=0;y<width;y++){
                        fputc(image_r[x][y],f_out);
                        fputc(image_g[x][y],f_out);
                        fputc(image_b[x][y],f_out);
                }
        }
        printf("データは正しく出力されました。¥n");
        printf("..........................................¥n");

}



int main(void)
{
   std::ofstream ofs( "test.txt" );
        char file_name1[255]={0};
        char file_name[255]={0};
        char file_zero[255]={0};
        unsigned char (*image_r)[width];
        unsigned char (*image_g)[width];
    unsigned char (*image_b)[width];


        double (*y)[width];
        double (*u)[width];
        double (*v)[width];


    double ***ys;
        double ***yt;
        double ***yn;
        double ***yk;
        double ***yr;



        unsigned char (*r)[width];
        unsigned char (*g)[width];
        unsigned char (*b)[width];

        double (*block)[block_size_y];
        char t[100]={0},yesno[5]={0};
        int i,j,kka,k,ip,jp;
        int nbb[5]={0};
        FILE *fd1;
        FILE *f_out;
        ys=(double ***) malloc( sizeof(double **)*high);
        if (ys==NULL) exit(1);
        for (i=0;i<high;i++){
                ys[i]=(double**)malloc(sizeof(double*)*width);
                if (ys[i]==NULL) exit(1);
```

```
        for (j=0;j<width; j++){
                ys[i][j]=(double *)malloc(sizeof(double)*kk);
                if (ys[i][j]==NULL) exit(1);
        }
}
yr=(double ***) malloc( sizeof(double **)*high);
if (yr==NULL) exit(1);
for (i=0;i<high;i++){
        yr[i]=(double**)malloc(sizeof(double*)*width);
        if (yr[i]==NULL) exit(1);
        for (j=0;j<width; j++){
                yr[i][j]=(double *)malloc(sizeof(double)*kk);
                if (yr[i][j]==NULL) exit(1);
        }
}
yt=(double ***) malloc( sizeof(double **)*high);
if (yt==NULL) exit(1);
for (i=0;i<high;i++){
        yt[i]=(double**)malloc(sizeof(double*)*width);
        if (yt[i]==NULL) exit(1);
        for (j=0;j<width; j++){
                yt[i][j]=(double *)malloc(sizeof(double)*kk);
                if (yt[i][j]==NULL) exit(1);
        }
}
yn=(double ***) malloc( sizeof(double **)*high);
if (yn==NULL) exit(1);
for (i=0;i<high;i++){
        yn[i]=(double**)malloc(sizeof(double*)*width);
        if (yn[i]==NULL) exit(1);
        for (j=0;j<width; j++){
                yn[i][j]=(double *)malloc(sizeof(double)*kk);
                if (yn[i][j]==NULL) exit(1);
        }
}
yk=(double ***) malloc( sizeof(double **)*high);
if (yk==NULL) exit(1);
for (i=0;i<high;i++){
        yk[i]=(double**)malloc(sizeof(double*)*width);
        if (yk[i]==NULL) exit(1);
        for (j=0;j<width; j++){
                yk[i][j]=(double *)malloc(sizeof(double)*kk);
                if (yk[i][j]==NULL) exit(1);
        }
}


for(i=0; i<high; i++){
        for(j=0; j<width; j++){
                for(k=0; k<kk; k++){
                        ys[i][j][k] = 0;
                        yn[i][j][k] = 0;
                        yr[i][j][k] = 0;
                        yt[i][j][k] = 0;
```

```
                                        yk[i][j][k] = 0;
                                }
                        }
                }

                for(kka=0;kka<8;kka++){
                        printf("kk=%d¥n",kka);
                        t[0]=kka+48;
                        t[1]='.';
                        t[2]='p';
                        t[3]='p';
                        t[4]='m';
                        ///file_name1=file_zero;
                        strcpy(file_name1, "");
                        strcat(file_name1,t);
                        printf("22:%s::%s¥n",t,file_name1);
                           /* 文字の連結 */
                                        fd1= fopen (file_name1,"rb");
                                        if(NULL == fd1){
                                                printf("1/(2)に次の番号のファイルは存在しないの
で読込終了:%s¥n",file_name1);
                                                //return 0;
                                        }
                                        //printf("r_1_kk=%d¥n",kk);
                                        r=(unsigned char
(*)[width])calloc(high*width,sizeof(unsigned char));
                                        if(r==NULL)
                                                printf("faild¥n");
                                        else ;
                                        g=(unsigned char
(*)[width])calloc(high*width,sizeof(unsigned char));
                                        if(g==NULL)
                                                printf("faild¥n");
                                        else ;
                                        b=(unsigned char
(*)[width])calloc(high*width,sizeof(unsigned char));
                                        if(b==NULL)
                                                printf("faild¥n");
                                        else ;
                                        getc(fd1);getc(fd1);
                                        pbm_getint(fd1);pbm_getint(fd1);pbm_getint(fd1);
                                        read_ppm(fd1,r,g,b);
                                        i=0,j=0,ip=199,jp=199;
                                        printf("r[%d][%d][%d]=%d¥n", ip+j, jp+i, kka+0,
r[ip+j][jp+i]);
                                        printf("g[%d][%d][%d]=%d¥n", ip+j, jp+i, kka+0,
g[ip+j][jp+i]);
                                        printf("b[%d][%d][%d]=%d¥n", ip+j, jp+i, kka+0,
b[ip+j][jp+i]);
                                        //printf("read_ppm_kk=%d¥n",kk);

                                        y=(double (*)[width])calloc(high*width,sizeof(double));

                                        if(y==NULL)
```

```
                                        printf("faild¥n");
                                else ;

                                u=(double (*)[width])calloc(high*width,sizeof(double));
                                if(u==NULL)
                                        printf("ptemp_faild¥n");
                                else ;

                                v=(double (*)[width])calloc(high*width,sizeof(double));
                                if(v==NULL)
                                        printf("ptemp_faild¥n");
                                else ;
                                // printf("ptemp_ok¥n");
                                //   printf("r[%d][%d][%d]=%df¥n", ip+j, jp+i, kka+0,
r[ip+j][jp+i]);
                                // printf("g[%d][%d][%d]=%d¥n", ip+j, jp+i, kka+0,
g[ip+j][jp+i]);
                                // printf("b[%d][%d][%d]=%d¥n", ip+j, jp+i, kka+0,
b[ip+j][jp+i]);
                                Y_U_V(r,g,b,y,u,v);
                                ///printf("Y_U_V_kk=%d¥n",kk);
                                i=0,j=0,ip=199,jp=199;
                                // printf("y[%d][%d][%d]=%f¥n", ip+j, jp+i, kka+0,
y[ip+j][jp+i]);
                                //printf("u[%d][%d][%d]=%f¥n", ip+j, jp+i, kka+0,
u[ip+j][jp+i]);
                                // printf("v[%d][%d][%d]=%f¥n", ip+j, jp+i, kka+0,
v[ip+j][jp+i]);
                                free(r);
                                free(g);
                                free(b);
                                //yi=(double *)malloc(high*width*sizeof(double));
                                ///yr_temp=(double *)malloc(high*width*sizeof(double));
//追加 2 次 メモリを確保します
                                k=0;
                                for(i=0;i<high;i++){
                                        for(j=0;j<width;j++){
                                                ys[i][j][kka]=y[i][j];    /// 追加 3d
                                                k++;
                        }
                                }
                                printf("end of kk=%d¥n",kk);
                                /// scanf("%s",file_zero);
                                //  yr=double&*y_temp[(299+i)*(199+j)];
                                //printf("y_temp=%f,:%f::%f::%f¥n¥n
"y_temp[jp+i][ip+j],y_temp[jp+i+1][ip+j],y_temp[jp+i+2][ip+j],y_temp[jp+i+3][ip+j]);
                                // DCT 変換
                                printf("cos 変換 ¥n");
                                //scanf("%s",file_name);
                                i=0,j=0,ip=199,jp=199;
                                printf("ys[%d][%d][%d][%f]=%f¥n", ip+j, jp+i, kka,
ys[ip+j][jp+i][kka],ys[ip+j][jp+i][kka]);

                                for(j=0;j<8;j++){
```

```
                                                yn[ip+j][jp+i][kka]=
ys[ip+j][jp+i][kka]*0.3536+ys[ip+j][jp+i+1][kka]*0.3536    +ys[ip+j][jp+i+2][kka]*0.3536
+ys[ip+j][jp+i+3][kka]*0.3536    +ys[ip+j][jp+i+4][kka]*0.3536    +ys[ip+j][jp+i+5][kka]*0.3536
+ys[ip+j][jp+i+6][kka]*0.3536    +ys[ip+j][jp+i+7][kka]*0.3536; //埋込みパターン

          yn[ip+j][jp+i+1][kka]=ys[ip+j][jp+i][kka]*0.4904+ys[ip+j][jp+i+1][kka]*0.4157
+ys[ip+j][jp+i+2][kka]*0.2778    +ys[ip+j][jp+i+3][kka]*0.0976
+ys[ip+j][jp+i+4][kka]*(-0.0976)+ys[ip+j][jp+i+5][kka]*(-0.2778)+ys[ip+j][jp+i+6][kka]*(-0.415
7)+ys[ip+j][jp+i+7][kka]*(-0.4904);

          yn[ip+j][jp+i+2][kka]=ys[ip+j][jp+i][kka]*0.4632+ys[ip+j][jp+i+1][kka]*0.1913
+ys[ip+j][jp+i+2][kka]*(-0.1913)+ys[ip+j][jp+i+3][kka]*(-0.4632)+ys[ip+j][jp+i+4][kka]*(-0.463
2)+ys[ip+j][jp+i+5][kka]*(-0.1913)+ys[ip+j][jp+i+6][kka]*0.1913
+ys[ip+j][jp+i+7][kka]*0.4632;

          yn[ip+j][jp+i+3][kka]=ys[ip+j][jp+i][kka]*0.4157+ys[ip+j][jp+i+1][kka]*(-0.0976)+ys[i
p+j][jp+i+2][kka]*(-0.4904)+ys[ip+j][jp+i+3][kka]*(-0.2778)+ys[ip+j][jp+i+4][kka]*0.2778
+ys[ip+j][jp+i+5][kka]*0.4904    +ys[ip+j][jp+i+6][kka]*(0.0976)
+ys[ip+j][jp+i+7][kka]*(-0.4157);

          yn[ip+j][jp+i+4][kka]=ys[ip+j][jp+i][kka]*0.3536+ys[ip+j][jp+i+1][kka]*(-0.3536)+ys[i
p+j][jp+i+2][kka]*(-0.3536)+ys[ip+j][jp+i+3][kka]*0.3536    +ys[ip+j][jp+i+4][kka]*0.3536
+ys[ip+j][jp+i+5][kka]*(-0.3536)+ys[ip+j][jp+i+6][kka]*(-0.3536)+ys[ip+j][jp+i+7][kka]*0.3536;

          yn[ip+j][jp+i+5][kka]=ys[ip+j][jp+i][kka]*0.2778+ys[ip+j][jp+i+1][kka]*(-0.4904)+ys[i
p+j][jp+i+2][kka]*(0.0976) +ys[ip+j][jp+i+3][kka]*0.4157
+ys[ip+j][jp+i+4][kka]*(-0.4157)+ys[ip+j][jp+i+5][kka]*(-0.0976)+ys[ip+j][jp+i+6][kka]*0.4904
+ys[ip+j][jp+i+7][kka]*(-0.2778);

          yn[ip+j][jp+i+6][kka]=ys[ip+j][jp+i][kka]*0.1913+ys[ip+j][jp+i+1][kka]*(-0.4632)+ys[i
p+j][jp+i+2][kka]*0.4632
+ys[ip+j][jp+i+3][kka]*(-0.1913)+ys[ip+j][jp+i+4][kka]*(-0.1913)+ys[ip+j][jp+i+5][kka]*0.4632
+ys[ip+j][jp+i+6][kka]*(-0.4632)+ys[ip+j][jp+i+7][kka]*0.1913;

          yn[ip+j][jp+i+7][kka]=ys[ip+j][jp+i][kka]*0.0976+ys[ip+j][jp+i+1][kka]*(-0.2778)+ys[i
p+j][jp+i+2][kka]*(0.4157)
+ys[ip+j][jp+i+3][kka]*(-0.4904)+ys[ip+j][jp+i+4][kka]*0.4904+ys[ip+j][jp+i+5][kka]*(-0.4157)
+ys[ip+j][jp+i+6][kka]*0.2778    +ys[ip+j][jp+i+7][kka]*(-0.0976);
                                                printf("1 変換
yn=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yn[ip+j][jp+i][kka],yn[ip+j][jp+i+1][kka],yn[ip+j][jp+i+2][kka],yn[ip+j][jp+i+3][kka],yn[ip+
j][jp+i+4][kka],yn[ip+j][jp+i+5][kka],yn[ip+j][jp+i+6][kka],yn[ip+j][jp+i+7][kka]);
                                                //ofs << "1 変換:" <<
yn[ip+j][jp+i][kka]<<":*:"<<yn[ip+j][jp+i+1][kka]
<<":*:"<<yn[ip+j][jp+i+2][kka]<<":*:"<<yn[ip+j][jp+i+3][kka]<<":*:"<<yn[ip+j][jp+i+4][kka]<<":
*:"<<yn[ip+j][jp+i+5][kka]<<":*:"<<yn[ip+j][jp+i+6][kka]<<":*:"<<yn[ip+j][jp+i+7][kka]<<std::e
ndl;
                                                ofs << "1d 原画像:"<<"kka="<< kka<<":*:" <<
ys[ip+j][jp+i][kka]<<":*:"<<ys[ip+j][jp+i+1][kka]<<":*:"<<ys[ip+j][jp+i+2][kka]<<":*:"<<ys[ip+
j][jp+i+3][kka]<<":*:"<<ys[ip+j][jp+i+4][kka]<<":*:"<<ys[ip+j][jp+i+5][kka]<<":*:"<<ys[ip+j][j
p+i+6][kka]<<":*:"<<ys[ip+j][jp+i+7][kka]<<std::endl;
                                                ofs << "1d 変換:" <<"kka="<<
kka<<":*:"<<yn[ip+j][jp+i][kka]<<":*:"<<yn[ip+j][jp+i+1][kka]<<":*:"<<yn[ip+j][jp+i+2][kka]<<"
:*:"<<yn[ip+j][jp+i+3][kka]<<":*:"<<yn[ip+j][jp+i+4][kka]<<":*:"<<yn[ip+j][jp+i+5][kka]<<":*:"
```

```
<<        yn[ip+j][jp+i+6][kka]<<":*:"<<yn[ip+j][jp+i+7][kka]<<std::endl<<std::endl;
                                        }


                                        //// printf("yn[%d][%d][%d][%f]=%f¥n", ip+j, jp+i, kka,
yn[ip+j][jp+i][kka],yn[ip+j][jp+i][kka]);
                                        // 転置変換
                                        for (i=0;i<8;i++){
                                                for(j=0;j<8;j++){

          yr[ip+i][jp+j][kka]=yn[ip+j][jp+i][kka];
                                                                //
printf("yr=%f,¥ ",yr[ip+i][jp+j][kka]);
                                                                ofs << "転置変換:" <<
yr[ip+i][jp+j][kka]<<":*:"<<std::endl;
                                                }
                                        }
                                        // 転置変換終了
                                        //  2 次変換
                                        for(j=0;j<8;j++){
                                                i=0;
                                                yt[ip+j][jp+i][kka]=
yr[ip+j][jp+i][kka]*0.3536+yr[ip+j][jp+i+1][kka]*0.3536   +yr[ip+j][jp+i+2][kka]*0.3536
+yr[ip+j][jp+i+3][kka]*0.3536   +yr[ip+j][jp+i+4][kka]*0.3536   +yr[ip+j][jp+i+5][kka]*0.3536
+yr[ip+j][jp+i+6][kka]*0.3536   +yr[ip+j][jp+i+7][kka]*0.3536; //埋込みパターン

          yt[ip+j][jp+i+1][kka]=yr[ip+j][jp+i][kka]*0.4904+yr[ip+j][jp+i+1][kka]*0.4157
+yr[ip+j][jp+i+2][kka]*0.2778   +yr[ip+j][jp+i+3][kka]*0.0976
+yr[ip+j][jp+i+4][kka]*(-0.0976)+yr[ip+j][jp+i+5][kka]*(-0.2778)+yr[ip+j][jp+i+6][kka]*(-0.415
7)+yr[ip+j][jp+i+7][kka]*(-0.4904);

          yt[ip+j][jp+i+2][kka]=yr[ip+j][jp+i][kka]*0.4632+yr[ip+j][jp+i+1][kka]*0.1913
+yr[ip+j][jp+i+2][kka]*(-0.1913)+yr[ip+j][jp+i+3][kka]*(-0.4632)+yr[ip+j][jp+i+4][kka]*(-0.463
2)+yr[ip+j][jp+i+5][kka]*(-0.1913)+yr[ip+j][jp+i+6][kka]*0.1913
+yr[ip+j][jp+i+7][kka]*0.4632;

          yt[ip+j][jp+i+3][kka]=yr[ip+j][jp+i][kka]*0.4157+yr[ip+j][jp+i+1][kka]*(-0.0976)+yr[i
p+j][jp+i+2][kka]*(-0.4904)+yr[ip+j][jp+i+3][kka]*(-0.2778)+yr[ip+j][jp+i+4][kka]*0.2778
+yr[ip+j][jp+i+5][kka]*0.4904   +yr[ip+j][jp+i+6][kka]*(0.0976)
+yr[ip+j][jp+i+7][kka]*(-0.4157);

          yt[ip+j][jp+i+4][kka]=yr[ip+j][jp+i][kka]*0.3536+yr[ip+j][jp+i+1][kka]*(-0.3536)+yr[i
p+j][jp+i+2][kka]*(-0.3536)+yr[ip+j][jp+i+3][kka]*0.3536   +yr[ip+j][jp+i+4][kka]*0.3536
+yr[ip+j][jp+i+5][kka]*(-0.3536)+yr[ip+j][jp+i+6][kka]*(-0.3536)+yr[ip+j][jp+i+7][kka]*0.3536;

          yt[ip+j][jp+i+5][kka]=yr[ip+j][jp+i][kka]*0.2778+yr[ip+j][jp+i+1][kka]*(-0.4904)+yr[i
p+j][jp+i+2][kka]*(0.0976) +yr[ip+j][jp+i+3][kka]*0.4157
+yr[ip+j][jp+i+4][kka]*(-0.4157)+yr[ip+j][jp+i+5][kka]*(-0.0976)+yr[ip+j][jp+i+6][kka]*0.4904
+yr[ip+j][jp+i+7][kka]*(-0.2778);

          yt[ip+j][jp+i+6][kka]=yr[ip+j][jp+i][kka]*0.1913+yr[ip+j][jp+i+1][kka]*(-0.4632)+yr[i
p+j][jp+i+2][kka]*0.4632
+yr[ip+j][jp+i+3][kka]*(-0.1913)+yr[ip+j][jp+i+4][kka]*(-0.1913)+yr[ip+j][jp+i+5][kka]*0.4632
+yr[ip+j][jp+i+6][kka]*(-0.4632)+yr[ip+j][jp+i+7][kka]*0.1913;
```

```
        yt[ip+j][jp+i+7][kka]=yr[ip+j][jp+i][kka]*0.0976+yr[ip+j][jp+i+1][kka]*(-0.2778)+yr[i
p+j][jp+i+2][kka]*(0.4157) +yr[ip+j][jp+i+3][kka]*(-0.4904)+yr[ip+j][jp+i+4][kka]*0.4904
+yr[ip+j][jp+i+5][kka]*(-0.4157)+yr[ip+j][jp+i+6][kka]*0.2778
+yr[ip+j][jp+i+7][kka]*(-0.0976);
                                                printf("2***変換
yt=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yt[ip+j][jp+i][kka],yt[ip+j][jp+i+1][kka],yt[ip+j][jp+i+2][kka],yt[ip+j][jp+i+3][kka],yt[ip+
j][jp+i+4][kka],yt[ip+j][jp+i+5][kka],yt[ip+j][jp+i+6][kka],yt[ip+j][jp+i+7][kka]);
                                                ofs << "2d 原画像:" <<"kka="<< kka<<":*:"<<
yr[ip+j][jp+i][kka]<<":*:"<<yr[ip+j][jp+i+1][kka]<<":*:"<<yr[ip+j][jp+i+2][kka]<<":*:"<<yr[ip+
j][jp+i+3][kka]<<":*:"<<yr[ip+j][jp+i+4][kka]<<":*:"<<yr[ip+j][jp+i+5][kka]<<":*:"<<yr[ip+j][j
p+i+6][kka]<<":*:"<<yr[ip+j][jp+i+7][kka]<<std::endl;
                                                ofs << "2d***変換:" <<"kka="<< kka<<":*:"<<
yt[ip+j][jp+i][kka]<<":*:"<<yt[ip+j][jp+i+1][kka]<<":*:"<<yt[ip+j][jp+i+2][kka]<<":*:"<<yt[ip+
j][jp+i+3][kka]<<":*:"<<yt[ip+j][jp+i+4][kka]<<":*:"<<yt[ip+j][jp+i+5][kka]<<":*:"<<yt[ip+j][j
p+i+6][kka]<<":*:"<<yt[ip+j][jp+i+7][kka]<<std::endl<<std::endl;
                                        }
                                        // printf("2 原画像 y=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yr[ip+j][jp+i],yr[ip+j][jp+i+1],yr[ip+j][jp+i+2],yr[ip+j][jp+i+3],yr[ip+j][jp+i+4],yr[ip+j][
jp+i+5],yr[ip+j][jp+i+6],yr[ip+j][jp+i+7]);
                                        // printf("変換 y=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",y[ip+j][jp+i],y[ip+j][jp+i+1],y[ip+j][jp+i+2],y[ip+j][jp+i+3],y[ip+j][jp+i+4],y[ip+j][jp+i+5
],y[ip+j][jp+i+6],y[ip+j][jp+i+7]);
                                        // ofs << "2d 変換:" <<
yt[ip+j][jp+i][kka]<<":*:"<<yt[ip+j][jp+i+1][kka]<<":*:"<<yt[ip+j][jp+i+2][kka]<<":*:"<<yt[ip+
j][jp+i+3][kka]<<":*:"<<yt[ip+j][jp+i+4][kka]<<":*:"<<yt[ip+j][jp+i+5][kka]<<":*:"<<yt[ip+j][j
p+i+6][kka]<<":*:"<<yt[ip+j][jp+i+7][kka]<<std::endl;
}  /// 連番終了
//2 変換終了
/// 3d 変換
ip=199;jp=199;kka=0;
   for(j=0;j<8;j++){
            for(i=0;i<8;i++){

                        printf("3d 変換 yt2=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yt[ip+j][jp+i][kka],yt[ip+j][jp+i][kka+1],yt[ip+j][jp+i][kka+2],yt[ip+j][jp+i][kka+3],yt[ip+
j][jp+i][kka+4], yt[ip+j][jp+i][kka+5],yt[ip+j][jp+i][kka+6],yt[ip+j][jp+i][kka+7]);
                        yk[ip+j][jp+i][kka]=
yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i][kka+1]*0.3536   +yt[ip+j][jp+i][kka+2]*0.3536
+yt[ip+j][jp+i][kka+3]*0.3536   +yt[ip+j][jp+i][kka+4]*0.3536   +yt[ip+j][jp+i][kka+5]*0.3536
+yt[ip+j][jp+i][kka+6]*0.3536   +yt[ip+j][jp+i][kka+7]*0.3536; //埋込みパターン

yk[ip+j][jp+i][kka+1]=yt[ip+j][jp+i][kka]*0.4904+yt[ip+j][jp+i][kka+1]*0.4157
+yt[ip+j][jp+i][kka+2]*0.2778   +yt[ip+j][jp+i][kka+3]*0.0976
+yt[ip+j][jp+i][kka+4]*(-0.0976)+yt[ip+j][jp+i][kka+5]*(-0.2778)+yt[ip+j][jp+i][kka+6]*(-0.415
7)+yt[ip+j][jp+i][kka+7]*(-0.4904);

yk[ip+j][jp+i][kka+2]=yt[ip+j][jp+i][kka]*0.4632+yt[ip+j][jp+i][kka+1]*0.1913
+yt[ip+j][jp+i][kka+2]*(-0.1913)+yt[ip+j][jp+i][kka+3]*(-0.4632)+yt[ip+j][jp+i][kka+4]*(-0.463
2)+yt[ip+j][jp+i][kka+5]*(-0.1913)+yt[ip+j][jp+i][kka+6]*0.1913
+yt[ip+j][jp+i][kka+7]*0.4632;

yk[ip+j][jp+i][kka+3]=yt[ip+j][jp+i][kka]*0.4157+yt[ip+j][jp+i][kka+1]*(-0.0976)+yt[ip+j][jp+i
```

```
][kka+2]*(-0.4904)+yt[ip+j][jp+i][kka+3]*(-0.2778)+yt[ip+j][jp+i][kka+4]*0.2778
+yt[ip+j][jp+i][kka+5]*0.4904   +yt[ip+j][jp+i][kka+6]*(0.0976)
+yt[ip+j][jp+i][kka+7]*(-0.4157);

yk[ip+j][jp+i][kka+4]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i][kka+1]*(-0.3536)+yt[ip+j][jp+i
][kka+2]*(-0.3536)+yt[ip+j][jp+i][kka+3]*0.3536   +yt[ip+j][jp+i][kka+4]*0.3536
+yt[ip+j][jp+i][kka+5]*(-0.3536)+yt[ip+j][jp+i][kka+6]*(-0.3536)+yt[ip+j][jp+i][kka+7]*0.3536;

yk[ip+j][jp+i][kka+5]=yt[ip+j][jp+i][kka]*0.2778+yt[ip+j][jp+i][kka+1]*(-0.4904)+yt[ip+j][jp+i
][kka+2]*(0.0976) +yt[ip+j][jp+i][kka+3]*0.4157
+yt[ip+j][jp+i][kka+4]*(-0.4157)+yt[ip+j][jp+i][kka+5]*(-0.0976)+yt[ip+j][jp+i][kka+6]*0.4904
+yt[ip+j][jp+i][kka+7]*(-0.2778);

yk[ip+j][jp+i][kka+6]=yt[ip+j][jp+i][kka]*0.1913+yt[ip+j][jp+i][kka+1]*(-0.4632)+yt[ip+j][jp+i
][kka+2]*0.4632
+yt[ip+j][jp+i][kka+3]*(-0.1913)+yt[ip+j][jp+i][kka+4]*(-0.1913)+yt[ip+j][jp+i][kka+5]*0.4632
+yt[ip+j][jp+i][kka+6]*(-0.4632)+yt[ip+j][jp+i][kka+7]*0.1913;

yk[ip+j][jp+i][kka+7]=yt[ip+j][jp+i][kka]*0.0976+yt[ip+j][jp+i][kka+1]*(-0.2778)+yt[ip+j][jp+i
][kka+2]*(0.4157)
+yt[ip+j][jp+i][kka+3]*(-0.4904)+yt[ip+j][jp+i][kka+4]*0.4904+yt[ip+j][jp+i][kka+5]*(-0.4157)
+yt[ip+j][jp+i][kka+6]*0.2778   +yt[ip+j][jp+i][kka+7]*(-0.0976);

                    printf("3d 変換 yk=%f,:%f::%f::%f::%f::%f::%f¥n
",yk[ip+j][jp+i][kka],yk[ip+j][jp+i][kka+1],yk[ip+j][jp+i][kka+2],yk[ip+j][jp+i][kka+3],yk[ip+
j][jp+i][kka+4], yk[ip+j][jp+i][kka+5],yk[ip+j][jp+i][kka+6],yk[ip+j][jp+i][kka+7]);
                    ofs << "3d 原画像:"
<<yt[ip+j][jp+i][kka]<<":*:"<<yt[ip+j][jp+i][kka+1]<<":*:"<<yt[ip+j][jp+i][kka+2]<<":*:"<<yt[i
p+j][jp+i][kka+3]<<":*:"<<yk[ip+j][jp+i][kka+4]<<":*:"<<
yt[ip+j][jp+i][kka+5]<<":*:"<<yt[ip+j][jp+i][kka+6]<<":*:"<<yt[ip+j][jp+i][kka+7]<<std::endl;
                    ofs << "3d 変換:" <<
yk[ip+j][jp+i][kka]<<":*:"<<yk[ip+j][jp+i][kka+1]<<":*:"<<yk[ip+j][jp+i][kka+2]<<":*:"<<yk[ip+
j][jp+i][kka+3]<<":*:"<<yk[ip+j][jp+i][kka+4]<<":*:"<<
yk[ip+j][jp+i][kka+5]<<":*:"<<yk[ip+j][jp+i][kka+6]<<":*:"<<yk[ip+j][jp+i][kka+7]<<std::endl;
          }
  }

  //量子化
  printf("量子化 ¥n");
  int po;po=32;
  ip=199;jp=199;
  for(kka=0;kka<2;kka++){
          for(j=0;j<2;j++){
                    i=1;
                    if(yk[ip+j][jp+i][kka]>po){

yk[ip+j][jp+i][kka]=(int((yk[ip+j][jp+i][kka]-po)/(po*2)+1))*(po*2);
                              printf("量子化 10 ¥n");
                    }
                    if(yk[ip+j][jp+i][kka]<-po){

yk[ip+j][jp+i][kka]=(int((yk[ip+j][jp+i][kka]+po)/(po*2)-1))*(po*2);
                              printf("量子化 20¥n");
                    }
```

```
                    if((yk[ip+j][jp+i][kka]<po)&&(yk[ip+j][jp+i][kka]>-po)){
                              yk[ip+j][jp+i][kka]=(int(yk[ip+j][jp+i][kka]/(po*2)))*(po*2);
                              printf("量子化 30 ¥n");
                    }
                    printf("y_01=%f,:kka=%d:¥n ",yk[ip+j][jp+i][kka],kka);
                    ofs <<"量子化:" << "kka=:"<<kka <<":*:" << yk[ip+j][jp+i][kka]<<std::endl;
          }
          i=0;j=0;
          if(yk[ip+j+1][jp+i][kka]>po){
                    yk[ip+j+1][jp+i][kka]=(int((yk[ip+j+1][jp+i][kka]-po)/(po*2)+1))*(po*2);
                    //printf("量子化 1 ¥n");
                    printf("y_01=%f,:kka=%d:¥n ",yk[ip+j+1][jp+i][kka],kka);
                    ofs <<"量子化:" << "kka=:"<<kka <<":*:" <<
yk[ip+j+1][jp+i][kka]<<std::endl;
          }
          if(yk[ip+j+1][jp+i][kka]<-po){
                    yk[ip+j+1][jp+i][kka]=(int((yk[ip+j+1][jp+i][kka]+po)/(po*2)-1))*(po*2);
                    //printf("量子化 2¥n");
                    printf("y_02=%f,:kka=%d:¥n ",yk[ip+j+1][jp+i][kka],kka);
                   ofs <<"量子化:" << "kka=:"<<kka <<":*:" << yk[ip+j+1][jp+i][kka]<<std::endl;
          }
          if((yk[ip+j+1][jp+i][kka]<po)&&(yk[ip+j+1][jp+i][kka]>-po)){
                    yk[ip+j+1][jp+i][kka]=(int(yk[ip+j+1][jp+i][kka]/(po*2)))*(po*2);
                    //printf("量子化 3 ¥n");
                    printf("y_03=%f,:kka=%d:¥n ",yk[ip+j+1][jp+i][kka],kka);
                    ofs <<"量子化:" << "kka=:"<<kka <<":*:" <<
yk[ip+j+1][jp+i][kka]<<std::endl;
          }
   }


   // printf("y_01=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yk[ip+i][jp+j][kka],yk[ip+i][jp+j][kka],yk[ip+i][jp+j][kka],yk[ip+i][jp+j][kka],yk[ip+i][jp+
j][kka],yk[ip+i][jp+j][kka],yk[ip+i][jp+j][kka],yk[ip+i][jp+j][kka]);
   //  ofs << "量子化変換 01:" <<
yk[ip+i][jp+j][kka]<<":*:"<<yk[ip+i][jp+j][kka]<<":*:"<<yk[ip+i][jp+j][kka]<<":*:"<<yk[ip+i][j
p+j][kka]<<":*:"<<yk[ip+i][jp+j][kka]<<":*:"<<yk[ip+i][jp+j][kka]<<":*:"<<yk[ip+i][jp+j][kka]<
<":*:"<<yk[ip+i][jp+j][kka]<<std::endl<<std::endl;
   //dct--3d 逆変換
   printf("dct--3d 逆変換 ¥n");
   //for(kka=0;kka<8;kka++){
   ip=199;jp=199;kka=0;
   for(j=0;j<8;j++){
          for(i=0;i<8;i++){
                    yt[ip+j][jp+i][kka]=
yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*0.4904   +yk[ip+j][jp+i][kka+2]*0.4632
+yk[ip+j][jp+i][kka+3]*0.4157   +yk[ip+j][jp+i][kka+4]*0.3536    +yk[ip+j][jp+i][kka+5]*0.2778
+yk[ip+j][jp+i][kka+6]*0.1913   +yk[ip+j][jp+i][kka+7]*0.0976; //埋込みパターン

yt[ip+j][jp+i][kka+1]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*0.4157
+yk[ip+j][jp+i][kka+2]*0.1913
+yk[ip+j][jp+i][kka+3]*(-0.0976)+yk[ip+j][jp+i][kka+4]*(-0.3536)
+yk[ip+j][jp+i][kka+5]*(-0.4904)+yk[ip+j][jp+i][kka+6]*(-0.4632)+yk[ip+j][jp+i][kka+7]*(-0.277
8);
```

```
yt[ip+j][jp+i][kka+2]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*0.2778
+yk[ip+j][jp+i][kka+2]*(-0.1913)
+yk[ip+j][jp+i][kka+3]*(-0.4904)+yk[ip+j][jp+i][kka+4]*(-0.3536)
+yk[ip+j][jp+i][kka+5]*(0.0976) +yk[ip+j][jp+i][kka+6]*0.4632
+yk[ip+j][jp+i][kka+7]*(0.4157);

yt[ip+j][jp+i][kka+3]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*0.0976
+yk[ip+j][jp+i][kka+2]*(-0.4904 )+yk[ip+j][jp+i][kka+3]*(-0.2778)+yk[ip+j][jp+i][kka+4]*0.3536
+yk[ip+j][jp+i][kka+5]*0.4157
+yk[ip+j][jp+i][kka+6]*(-0.1913)+yk[ip+j][jp+i][kka+7]*(-0.4904);

yt[ip+j][jp+i][kka+4]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*(-0.0976)+yk[ip+j][jp+i
][kka+2]*(-0.4904) +yk[ip+j][jp+i][kka+3]*0.2778   +yk[ip+j][jp+i][kka+4]*0.3536
+yk[ip+j][jp+i][kka+5]*(-0.4157)+yk[ip+j][jp+i][kka+6]*(-0.1913)+yk[ip+j][jp+i][kka+7]*0.4904;

yt[ip+j][jp+i][kka+5]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*(-0.2778)+yk[ip+j][jp+i
][kka+2]*(-0.1913) +yk[ip+j][jp+i][kka+3]*0.4904   +yk[ip+j][jp+i][kka+4]*(-0.3536)
+yk[ip+j][jp+i][kka+5]*(-0.0976)+yk[ip+j][jp+i][kka+6]*0.4632
+yk[ip+j][jp+i][kka+7]*(-0.4157);

yt[ip+j][jp+i][kka+6]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*(-0.4157)+yk[ip+j][jp+i
][kka+2]*(0.1913) +yk[ip+j][jp+i][kka+3]*(0.0976) +yk[ip+j][jp+i][kka+4]*(-0.3536)
+yk[ip+j][jp+i][kka+5]*0.4904   +yk[ip+j][jp+i][kka+6]*(-0.4632)+yk[ip+j][jp+i][kka+7]*0.2778;

yt[ip+j][jp+i][kka+7]=yk[ip+j][jp+i][kka]*0.3536+yk[ip+j][jp+i][kka+1]*(-0.4904)+yk[ip+j][jp+i
][kka+2]*(0.4904) +yk[ip+j][jp+i][kka+3]*(-0.4157)+yk[ip+j][jp+i][kka+4]*0.3536
+yk[ip+j][jp+i][kka+5]*(-0.2778)+yk[ip+j][jp+i][kka+6]*0.1913
+yk[ip+j][jp+i][kka+7]*(-0.0976);

                    printf("d3 変換 yt=%f,:%f::%f::%f::%f::%f::%f¥n
",yt[ip+j][jp+i][kka],yt[ip+j][jp+i][kka+1],yt[ip+j][jp+i][kka+2],yt[ip+j][jp+i][kka+3],yt[ip+
j][jp+i][kka+4], yt[ip+j][jp+i][kka+5],yt[ip+j][jp+i][kka+6],yt[ip+j][jp+i][kka+7]);
                    ofs << "d3 原画像:"
<<yk[ip+j][jp+i][kka]<<":*:"<<yk[ip+j][jp+i][kka+1]<<":*:"<<yk[ip+j][jp+i][kka+2]<<":*:"<<yk[i
p+j][jp+i][kka+3]<<":*:"<<yk[ip+j][jp+i][kka+4]<<":*:"<<
yk[ip+j][jp+i][kka+5]<<":*:"<<yk[ip+j][jp+i][kka+6]<<":*:"<<yt[ip+j][jp+i][kka+7]<<std::endl;
                    ofs << "d3 変換:" <<
yt[ip+j][jp+i][kka]<<":*:"<<yt[ip+j][jp+i][kka+1]<<":*:"<<yt[ip+j][jp+i][kka+2]<<":*:"<<yt[ip+
j][jp+i][kka+3]<<":*:"<<yt[ip+j][jp+i][kka+4]<<":*:"<<
yt[ip+j][jp+i][kka+5]<<":*:"<<yt[ip+j][jp+i][kka+6]<<":*:"<<yt[ip+j][jp+i][kka+7]<<std::endl<<
std::endl;
            }
   }
   // dct--2d 逆変換
   printf("dct--2d 逆変換 ¥n");
   for(kka=0;kka<8;kka++){
            for(j=0;j<8;j++){
                    i=0;
                    yr[ip+j][jp+i][kka]=
yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*0.4904+   yt[ip+j][jp+i+2][kka]*0.4632+
yt[ip+j][jp+i+3][kka]*0.4157+   yt[ip+j][jp+i+4][kka]*0.3536    +yt[ip+j][jp+i+5][kka]*0.2778+
yt[ip+j][jp+i+6][kka]*0.1913+   yt[ip+j][jp+i+7][kka]*0.0976; //埋込みパターン
```

```
yr[ip+j][jp+i+1][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*0.4157+
yt[ip+j][jp+i+2][kka]*0.1913 +
yt[ip+j][jp+i+3][kka]*(-0.0976)+yt[ip+j][jp+i+4][kka]*(-0.3536)
+yt[ip+j][jp+i+5][kka]*(-0.4904)+yt[ip+j][jp+i+6][kka]*(-0.4632)+yt[ip+j][jp+i+7][kka]*(-0.277
8);

yr[ip+j][jp+i+2][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*0.2778+
yt[ip+j][jp+i+2][kka]*(-0.1913)+
yt[ip+j][jp+i+3][kka]*(-0.4904)+yt[ip+j][jp+i+4][kka]*(-0.3536)
+yt[ip+j][jp+i+5][kka]*(0.0976)+ yt[ip+j][jp+i+6][kka]*0.4632+
yt[ip+j][jp+i+7][kka]*(0.4157);

yr[ip+j][jp+i+3][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*0.0976+
yt[ip+j][jp+i+2][kka]*(-0.4904 )+yt[ip+j][jp+i+3][kka]*(-0.2778)+yt[ip+j][jp+i+4][kka]*0.3536
+yt[ip+j][jp+i+5][kka]*0.4157 +
yt[ip+j][jp+i+6][kka]*(-0.1913)+yt[ip+j][jp+i+7][kka]*(-0.4904);

yr[ip+j][jp+i+4][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*(-0.0976)+yt[ip+j][jp+i
+2][kka]*(-0.4904)+ yt[ip+j][jp+i+3][kka]*0.2778+   yt[ip+j][jp+i+4][kka]*0.3536
+yt[ip+j][jp+i+5][kka]*(-0.4157)+yt[ip+j][jp+i+6][kka]*(-0.1913)+yt[ip+j][jp+i+7][kka]*0.4904;

yr[ip+j][jp+i+5][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*(-0.2778)+yt[ip+j][jp+i
+2][kka]*(-0.1913)+ yt[ip+j][jp+i+3][kka]*0.4904+   yt[ip+j][jp+i+4][kka]*(-0.3536)
+yt[ip+j][jp+i+5][kka]*(-0.0976)+yt[ip+j][jp+i+6][kka]*0.4632+
yt[ip+j][jp+i+7][kka]*(-0.4157);

yr[ip+j][jp+i+6][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*(-0.4157)+yt[ip+j][jp+i
+2][kka]*(0.1913)+ yt[ip+j][jp+i+3][kka]*(0.0976)+  yt[ip+j][jp+i+4][kka]*(-0.3536)
+yt[ip+j][jp+i+5][kka]*0.4904+   yt[ip+j][jp+i+6][kka]*(-0.4632)+yt[ip+j][jp+i+7][kka]*0.2778;

yr[ip+j][jp+i+7][kka]=yt[ip+j][jp+i][kka]*0.3536+yt[ip+j][jp+i+1][kka]*(-0.4904)+yt[ip+j][jp+i
+2][kka]*(0.4904)+ yt[ip+j][jp+i+3][kka]*(-0.4157)+ yt[ip+j][jp+i+4][kka]*0.3536
+yt[ip+j][jp+i+5][kka]*(-0.2778)+yt[ip+j][jp+i+6][kka]*0.1913+
yt[ip+j][jp+i+7][kka]*(-0.0976);
        ofs << "d2 原画像:" <<"kka="<< kka<<":*:"<<
yt[ip+j][jp+i][kka]<<":*:"<<yt[ip+j][jp+i+1][kka]<<":*:"<<yt[ip+j][jp+i+2][kka]<<":*:"<<yt[ip+
j][jp+i+3][kka]<<":*:"<<yt[ip+j][jp+i+4][kka]<<":*:"<<yt[ip+j][jp+i+5][kka]<<":*:"<<yt[ip+j][j
p+i+6][kka]<<":*:"<<yt[ip+j][jp+i+7][kka]<<std::endl;
        ofs << "d2 变换:" <<"kka="<< kka<<":*:"<<
yr[ip+j][jp+i][kka]<<":*:"<<yr[ip+j][jp+i+1][kka]<<":*:"<<yr[ip+j][jp+i+2][kka]<<":*:"<<yr[ip+
j][jp+i+3][kka]<<":*:"<<yr[ip+j][jp+i+4][kka]<<":*:"<<yr[ip+j][jp+i+5][kka]<<":*:"<<yr[ip+j][j
p+i+6][kka]<<":*:"<<yr[ip+j][jp+i+7][kka]<<std::endl<<std::endl;
        printf("1 变换 yr=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yr[ip+j][jp+i][kka],yr[ip+j][jp+i+1][kka],yr[ip+j][jp+i+2][kka],yr[ip+j][jp+i+3][kka],yr[ip+
j][jp+i+4][kka],yr[ip+j][jp+i+5][kka],yr[ip+j][jp+i+6][kka],yr[ip+j][jp+i+7][kka]);
            }

            // dct--2d---1d 逆变换
            printf(" dct--2d---1d 逆变换¥n");
            for (i=0;i<8;i++){
                    for(j=0;j<8;j++){
                            yn[ip+j][jp+i][kka]=yr[ip+i][jp+j][kka];
                            ofs << "d2---d1 变换:"<<":*:"<<yn[ip+j][jp+i][kka]<<std::endl;
                            printf("1 变换 yn=%f,¥ ",yn[ip+j][jp+i][kka]);
```

```
                    }
            }
            ////printf("1 変換 yn=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",yn[ip+j][jp+i][kka],yn[ip+j][jp+i+1],yn[ip+j][jp+2+i][kka],yn[ip+j][jp+3+i][kka],yn[ip+j][jp
+4+i][kka],yn[ip+j][jp+i+5][kka],yn[ip+j][jp+i+6][kka],yn[ip+j][jp+i+7][kka]);
            // dct---1d 逆変換
            printf("dct---1d 逆変換¥n");
            for(j=0;j<8;j++){
                    i=0;
                    ys[ip+j][jp+i][kka]=
yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*0.4904+   yn[ip+j][jp+2][kka]*0.4632+
yn[ip+j][jp+3][kka]*0.4157+   yn[ip+j][jp+4][kka]*0.3536    +yn[ip+j][jp+5][kka]*0.2778+
yn[ip+j][jp+6][kka]*0.1913+   yn[ip+j][jp+7][kka]*0.0976; //埋込みパターン

ys[ip+j][jp+i+1][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*0.4157+
yn[ip+j][jp+2][kka]*0.1913 +   yn[ip+j][jp+3][kka]*(-0.0976)+   yn[ip+j][jp+4][kka]*(-0.3536)
+yn[ip+j][jp+5][kka]*(-0.4904)+yn[ip+j][jp+6][kka]*(-0.4632)+yn[ip+j][jp+7][kka]*(-0.2778);

ys[ip+j][jp+i+2][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*0.2778+
yn[ip+j][jp+2][kka]*(-0.1913)+ yn[ip+j][jp+3][kka]*(-0.4904)+yn[ip+j][jp+4][kka]*(-0.3536)
+yn[ip+j][jp+5][kka]*(0.0976)+ yn[ip+j][jp+6][kka]*0.4632+   yn[ip+j][jp+7][kka]*(0.4157);

ys[ip+j][jp+i+3][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*0.0976+
yn[ip+j][jp+2][kka]*(-0.4904 )+yn[ip+j][jp+3][kka]*(-0.2778)+yn[ip+j][jp+4][kka]*0.3536
+yn[ip+j][jp+5][kka]*0.4157 +  yn[ip+j][jp+6][kka]*(-0.1913)+yn[ip+j][jp+7][kka]*(-0.4904);

ys[ip+j][jp+i+4][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*(-0.0976)+yn[ip+j][jp+2][kk
a]*(-0.4904)+ yn[ip+j][jp+3][kka]*0.2778+   yn[ip+j][jp+4][kka]*0.3536
+yn[ip+j][jp+5][kka]*(-0.4157)+yn[ip+j][jp+6][kka]*(-0.1913)+yn[ip+j][jp+7][kka]*0.4904;

ys[ip+j][jp+i+5][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*(-0.2778)+yn[ip+j][jp+2][kk
a]*(-0.1913)+  yn[ip+j][jp+3][kka]*0.4904+   yn[ip+j][jp+4][kka]*(-0.3536)
+yn[ip+j][jp+5][kka]*(-0.0976)+yn[ip+j][jp+6][kka]*0.4632+   yn[ip+j][jp+7][kka]*(-0.4157);

ys[ip+j][jp+i+6][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*(-0.4157)+yn[ip+j][jp+2][kk
a]*(0.1913)+ yn[ip+j][jp+3][kka]*(0.0976)+ yn[ip+j][jp+4][kka]*(-0.3536)
+yn[ip+j][jp+5][kka]*0.4904+   yn[ip+j][jp+6][kka]*(-0.4632)+yn[ip+j][jp+7][kka]*0.2778;

ys[ip+j][jp+i+7][kka]=yn[ip+j][jp][kka]*0.3536+yn[ip+j][jp+1][kka]*(-0.4904)+yn[ip+j][jp+2][kk
a]*(0.4904)+ yn[ip+j][jp+3][kka]*(-0.4157)+yn[ip+j][jp+4][kka]*0.3536
+yn[ip+j][jp+5][kka]*(-0.2778)+yn[ip+j][jp+6][kka]*0.1913+   yn[ip+j][jp+7][kka]*(-0.0976);
                    ofs << "d1 原画像:" <<"kka=" << kka<<":*:"<<
yn[ip+j][jp+i][kka]<<":*:"<<yn[ip+j][jp+i+1][kka]<<":*:"<<yn[ip+j][jp+i+2][kka]<<":*:"<<yn[ip+
j][jp+i+3][kka]<<":*:"<<yn[ip+j][jp+i+4][kka]<<":*:"<<yn[ip+j][jp+i+5][kka]<<":*:"<<yn[ip+j][j
p+i+6][kka]<<":*:"<<yn[ip+j][jp+i+7][kka]<<std::endl;
                    ofs << "d1 変換:" << "kka=" << kka<<":*:"<<
ys[ip+j][jp+i][kka]<<":*:"<<ys[ip+j][jp+1+i][kka]<<":*:"<<ys[ip+j][jp+2+i][kka]<<":*:"<<ys[ip+
j][jp+3+i][kka]<<":*:"<<ys[ip+j][jp+4+i][kka]<<":*:"<<ys[ip+j][jp+5+i][kka]<<":*:"<<ys[ip+j][j
p+6+i][kka]<<":*:"<<ys[ip+j][jp+7+i][kka]<<std::endl<<std::endl;
                    printf("d1 変換 ys=%f,:%f::%f::%f::%f::%f::%f::%f¥n
",ys[ip+j][jp+i][kka],ys[ip+j][jp+i+1][kka],ys[ip+j][jp+2+i][kka],ys[ip+j][jp+3+i][kka],ys[ip+
j][jp+4+i][kka],ys[ip+j][jp+i+5][kka],ys[ip+j][jp+i+6][kka],ys[ip+j][jp+i+7][kka]);
            }
            //  printf("d1 変換 ys=%f,:%f::%f::%f::%f::%f::%f::%f¥n
```

```
",ys[ip+j][jp+i][kka],ys[ip+j][jp+i+1],ys[ip+j][jp+2+i][kka],ys[ip+j][jp+3+i][kka],ys[ip+j][jp
+4+i][kka],ys[ip+j][jp+i+5][kka],ys[ip+j][jp+i+6][kka],ys[ip+j][jp+i+7][kka]);
            //画像の保存
            for(i=0;i<high;i++){
                    for(j=0;j<width;j++){
                            y[i][j]=ys[i][j][kka] ;    /// 追加 3d
                    }
            }
            printf("出力ファイル名(*.ppm):");
            scanf("%s",file_name);
            f_out=fopen(file_name,"wb");
            save_image_data_head(f_out);
            image_r=(unsigned char (*)[width])calloc(high*width,sizeof(unsigned char));
            if(image_r==NULL)
                    printf("faild¥n");
            else ;
            image_g=(unsigned char (*)[width])calloc(high*width,sizeof(unsigned char));
            if(image_g==NULL)
                    printf("faild¥n");
            else ;
            image_b=(unsigned char (*)[width])calloc(high*width,sizeof(unsigned char));
            if(image_b==NULL)
                    printf("faild¥n");
            else ;
            //yuv_rgb(image_r,image_g,image_b,yi,u,v);
            yuv_rgb(image_r,image_g,image_b,y,u,v);
            printf("5");
            printf("6");
            printf("7");
            printf("8");
            kaki(f_out,image_r,image_g,image_b);
    }




    free(image_r);
    free(image_g);
    free(image_b);
    free(ys);
    free(yn);
    free(yr);
    free(yt);
    free(yk);
    free(y);
    free(u);
    free(v);
    fclose(fd1);
    fclose(f_out);

}
```