平成 13 年度

学士学位論文

大容量映像コンテンツの統一利用のための画像転送方式に関する研究

Study on the transfer method for seamless utilization of mass video contents

1020308　　中平 和友

指導教員　　島村 和典 教授

2002 年 2 月 8 日

高知工科大学 情報システム工学科

要 旨

大容量映像コンテンツの統一利用のための
画像転送方式に関する研究

中平 和友

現在，バックボーンおよびアクセス網の高速化により，情報ネットワークにおけるエンドユーザが利用できる帯域幅は日増しに拡大を続けている．今後，アクセス網の完全な光化，光多重技術の進歩によってさらなる高速化が期待されている．超高速ネットワークが利用可能になることで，様々な映像コンテンツの，ネットワーク経由の利用が本格化し，PC 系デバイスによる映像コンテンツの利用頻度が増加することが予想される．PC 系デバイスで映像コンテンツを扱う際，自然画像と CG のファイルフォーマットが共通であれば，表示処理系を単純化し，デスクトップ画面との高い親和性を実現できる．しかし，現状では映像の種類，利用用途に応じ様々なファイルフォーマットが存在し，その扱いが煩雑になっている．

本研究では，様々な映像コンテンツに共通して利用可能な映像基盤を構築し，その統一的な利用を可能にすることを目指す．その第一の問題解決として，RGB ベースの画像転送方式の検討を行い，その仕様に従って画像転送系を構築し，様々な解像度別に画像転送実験を行った．実験の結果，QVGA 程度の解像度なら，60fps 以上の表示速度が達成された．それ以上の解像度については，画像データ量が莫大であるため，転送処理能力がボトルネックとなり，十分な表示速度が達成されなかった．これは本研究で用いているファイルフォーマットに従った場合，転送される画像データ量が莫大なものであるためである．今後の課題として，高い解像度を維持可能な，処理負荷の軽い RGB ベースの圧縮処理の検討が挙げられる．

キーワード　超高速ネットワーク, 大容量映像コンテンツ, 統一的な利用, RGB ベースの画像転送

Abstract

Study on the transfer method for seamless utilization of mass video contents

Kazutomo NAKAHIRA

Recently, backbone of the Internet and access line have been broaden day by day. There are great hopes that the bandwidth is more broaden by fiber-optic access line and WDM technology. The ultra high-speed networks are making distribution of various video contents via network popular. It implies that PC-based usage of video contents is increasing. When you display video contents on the PC-based screens, if video images and CG have a common file format, then we can ensure to simplify the display system and to have an affinity to desktop screens. But they has many file formats and you can not use video contents seamlessly.

The goal of this study is to construct a common image infrastructure for seamless utilization about varied types of images. A first problem solving, we considered a RGB-based image transfer method and implemented image transfer prototype. We run experiment on above prototype that sender side applications transmit 2,000 images to receiver about QVGA-XGA resolution. From experimental results, we achieved a refresh rate over 60 fps in QVGA format. However, It is not achieved in higher resolution because of large amount of the raw data. Considering RGB-based data compression that can keep high resolution and low load is our future work.

key words ultra high-speed network, mass video contents, seamless utilization, RGB-based image transfer

目次

図目次

表目次

第 1 章

序論

本章では，最初に本研究の目的を示す．次いで背景として，ネットワークの高速化，アプリケーションの多様化について述べる．

1.1 研究の目的

本研究では，自然画像と CG のような人工画像を，用途を問わず共通して利用可能な映像基盤を構築し，その統一的な利用を可能にすることを目指す．その第一の問題解決として，RGB ベースの画像転送方式を提案する．これは TV 技術よりの映像規格から，PC ベースの映像規格への標準の移行といった意味合いをも包含している．

1.2 研究の背景

1.2.1 利用可能な回線速度の向上

総務省によると，2005 年にはインターネットのバックボーンに用いられる回線の伝達能力は 5Tbps 以上に達すると予測されている．さらに，アクセス網の高速化も顕著である．2001 年，日本国内においても廉価に xDSL によるインターネットへの高速アクセスを提供するサービスプロバイダが登場し，急速な普及をしている．総務省のデータによると，2001 年当初は 1 万人程度だったユーザ数が，2001 年末には 130 万人を突破している (図 1.1 参照) [1].

但し，x DSL はメタルケーブルでしか利用できず，アクセス網の光化により利用不可と

図 1.1　DSL 加入者の推移のグラフ

なる点，現状では下り最高速度で 8Mbps 程度の通信速度しか提供していない点から，次世代の超高速ネットワークとは呼べない．しかし，過渡的な技術といえど，一般家庭において常時接続環境を提供し，多くの人から高速回線の利用に対する注目を集めたという意義は大きい．一方の FTTH では，最大 100Mbps 程度の帯域が利用可能となっており，将来的にはギガビット級の超高速ネットワークを提供する事が見込まれている．

1.2.2　利用されるアプリケーションの増加，多種類化

高速な回線が利用可能となれば，電子メールや Web ブラウジングといった，テキストベースのインターネット利用は快適に行うことが可能となる．そして，容量の大きさゆえ，インターネット経由での流通が少なかったビデオコンテンツの取り扱いも現実的になってきている．これにより，様々な種類のコンテンツと，それを利用するためのアプリケーションが増加する事が予想される．

では，PC によるインターネット利用において，実際にどのような用途のアプリケーションが使われているか，総務省発表の平成十三年度通信白書より抜粋して図 1.2 に示す．

これを見れば，現在電子メールの利用率は圧倒的なものであり，次いで情報検索・入手といった，ウェブブラウジング系のカテゴリの利用頻度が高い事が分かる．

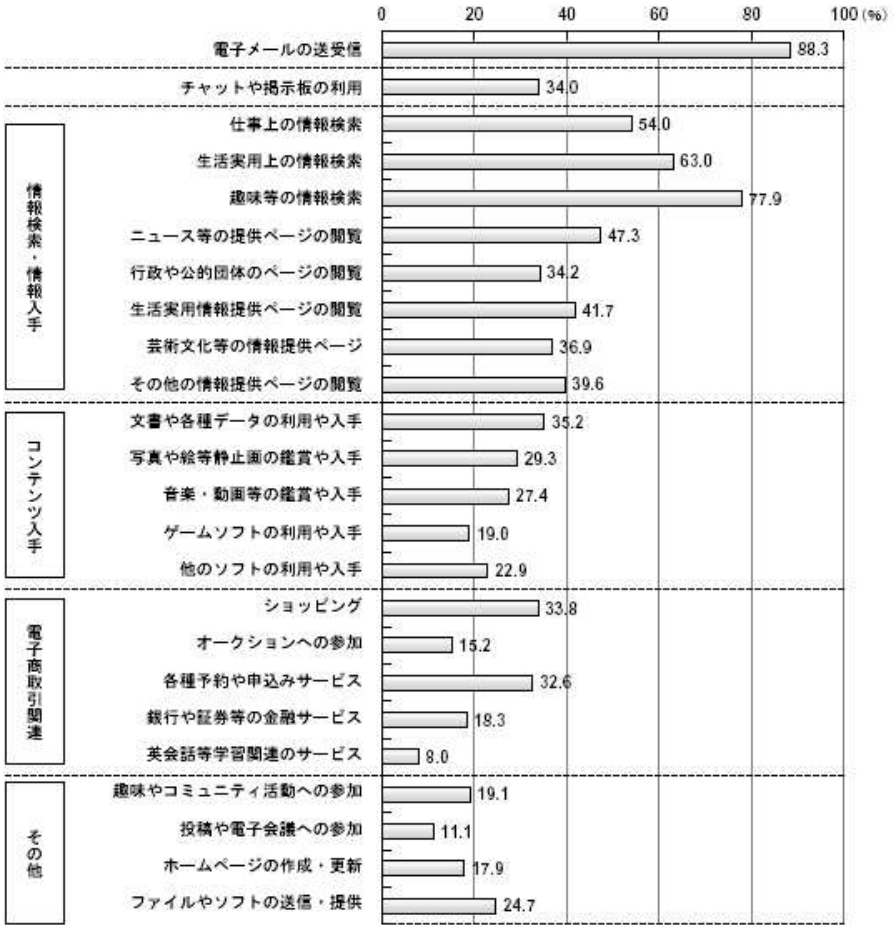

図 1.2　パソコンからインターネットを利用している人の利用用途（複数回答）

これらに比べればまだ少ないが，音楽，動画といったコンテンツの観賞や入手に利用していると答えた人の割合が，既に 27.4%にも及んでいることは注目すべき点である．現状において，既に様々なアプリケーションが利用されており，今後の回線速度の向上を考えれば，これらがさらに増えることは間違いないといえる．

1.3　既存の映像技術

現在，自然画像と CG は異なる処理系で扱われている．以下には，それぞれにおいて代表的な規格を取り上げ，その特徴や，利用されているアプリケーションの違いを挙げてゆく．

1.3.1　自然画像

画面の表示規格

自然画像を表現する典型的な例として，やはりアナログ TV が第一に挙げられる．その規格としては，日米で用いられている NTSC，西欧諸国の PAL などが挙げられる．両者は概ね同じ規格であるが，水平走査線数，リフレッシュレート等に違いがあるため，互換性がない．また，より高精細な映像を提供するために，HDTV 規格が考案されている．これらの規格の特徴を表 1.1 に簡単にまとめておく．

表 1.1

表示形式	水平走査線数	垂直操作周波数	アスペクト比
NTSC	525	60Hz	4:3
PAL	625	50Hz	4:3
HDTV	1125	60Hz	16:9

TV 電話などのアプリケーションを国際的に使うためには，世界共通のファールフォーマットが必要とされた．その結果として，共通フォーマットの CIF(Common Intermediate Format) と，その 4 分の 1 の画面サイズの QCIF が ITU H.261 で規定されている．この特徴を表 1.2 に示す．

表 1.2

表示形式	解像度 (pixel)	フレーム周波数	アスペクト比
CIF	352*288	最大 30Hz	4:3
QCIF	176*144	最大 30Hz	4:3

以上，各形式毎に様々な表示サイズ，リフレッシュレートが多数存在している．

色規格

前述のように様々な規格が存在する TV 系規格だが，色表現は共通して YUV 系の表現が使われている．具体的には以下の通りとなる．

TV では，YUV(Y=輝度信号，U=輝度成分と赤色信号との色差，V=輝度信号と青色信号との色差) 形式が採用されており，一般的に各信号の符号長には 8bit，256 階調が割り当てられている．また，人間の目が輝度成分に比べ色差成分に鈍感である事を利用し，色差信号の情報を削減する圧縮法が使われている．その中でよく使われているものとしては，YUV 4:2:2 と，YUV4:2:0 という方式がある．YUV 4:2:2 は，Y 信号のサンプリング間隔に対し，U，V 信号のサンプリング間隔を 2 倍広くとり，U，V データの個数を 2 分の 1 にすることで，全体の情報量を 3 分の 2 に削減する手法である (図 1.3 参照).

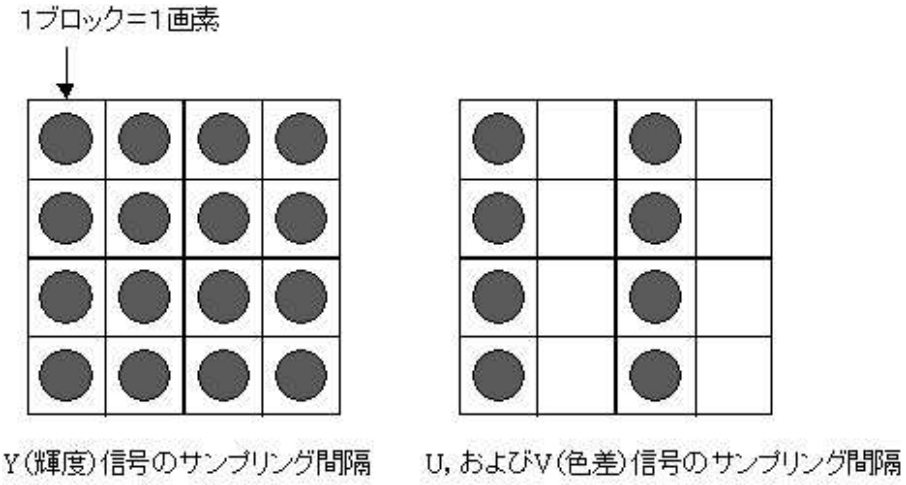

図 1.3　YUV 4:2:2 の画素サンプリング間隔

YUV 4:2:0 は，Y 信号のサンプリング間隔に対し，U，V 信号のサンプリング間隔を縦方向と横方向両方に 2 倍広くとり，U，V 信号のデータの個数を 4 分の 1 にすることで，全体の情報量を 2 分の 1 に削減する手法である (図 1.4 参照).

なお，実際に CRT へ映像を出力する際には，YUV 系から光の 3 原色である RGB 系への変換が必要となる．その変換式は以下に示すものが用いられている．

$R = Y + 1.402 \times V$

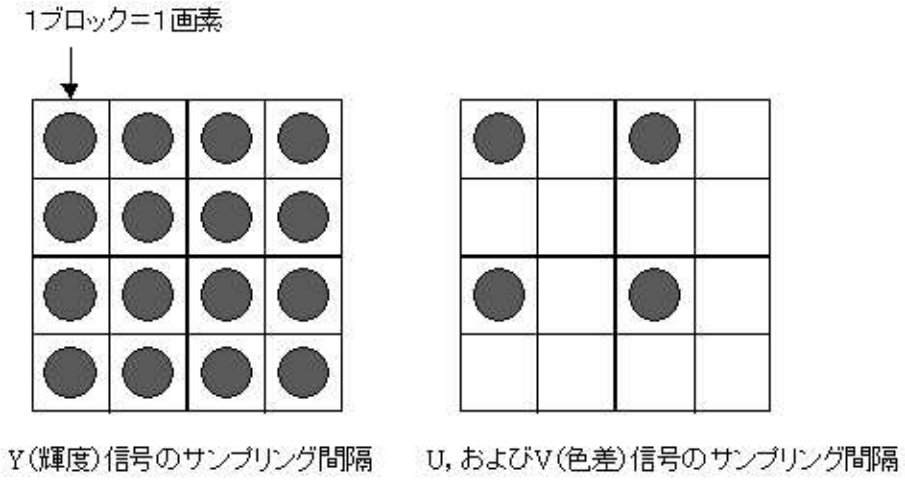

図 1.4　YUV 4:2:0 の画素サンプリング間隔

$$G = Y - 0.3441 \times U + 0.7139 \times V$$

$$B = Y + 1.7718 \times U - 0.0012 \times V$$

1.3.2　ディジタル映像規格

自然画像のディジタル符号化方式としては，MPEG が標準として広く普及している．画像データの符号化には，MPEG-1/2/4 を用いることができる．ここでは，その中から高画質な次世代ディジタル TV として期待されている MPEG-2 について注目する．

MPEG は，従来の TV 規格の影響を強く受けている点が多い．MPEG-2 では，プロファイルとレベルという指標により，フレームレートや解像度，その色表現が決定されている [4].

1.4　CG

CG は主に，PC 系デバイスでのディスプレイ上で利用するために作られた規格が多い．PC のデスクトップ画面も CG の一種である．今日普及している PC 系デバイスの大半では，色表現の規格に，CRT ディスプレイに表示されている画素と同じ，光の三原色である RGB 形式が用いられている．そのなかでも，RGB 各 8bit，計 24bit，16777216 色表示可

能なフルカラー規格がよく用いられている．また，解像度の規格に関しては，横×縦アスペクト比が 4:3 の，xGA 規格が用いられている．

その他，一般的に用いられている CG の規格については，2DCG 向けのものから 3DCG に至るまで実に様々な規格が氾濫しており，画像操作系のアプリケーションのベンダー毎に異なる規格が存在しているような状態にある [5].

1.5　本章のまとめ

以上のように, 映像コンテンツを扱う際, その用途, 映像の性質毎に様々なファイルフォーマットが存在している．また，それらの互換性が確保されておらず，映像同士の親和性に欠ける点や，利用時に各々に対応したアプリケーションを準備する必要があり，扱いが煩雑である点といった，解決すべき問題点がある．

もし，映像の表示，ネットワーク経由の送受信，ユーザ操作といった，映像コンテンツの利用が単一のアプリケーション上で統一して扱えるようになれば，上記のような問題は解決され，柔軟な映像利用が可能となる．

次章では，その実現のため，従来の映像形式に共通している要素を抽出し，それぞれの用途に利用可能な映像形式の検討を行う．

第 2 章

映像形式の検討

本章では，現状における自然画像，CG の取り扱いと，既存の技術の利点，問題点を参考に，共通利用に適したファイルフォーマットの提案を行う．

2.1 映像形式の現状

現在，自然画像と CG は異なる処理系が使われている．前述の通り，自然画像に関しては，MPEG が標準として普及しているが，映像形式が TV 規格ベースであり，PC 系デバイスのディスプレイへの表示及び CG との親和性に劣るという問題がある．CG に関しては，様々なファイル形式が無数に存在しており，ローカル環境においてはよく利用されているが，ネットワーク経由でローカル環境同様の高解像度，高リフレッシュレートで利用が可能な規格は存在していない．

しかし，これら映像コンテンツは，ディスプレイへの表示段階では同じく RGB 値の配列として扱われるという共通点を持つ．そこで，最終的な表示段階の形式，すなわち RGB ベースのファイルフォーマットを利用することで，自然画像，CG を問わず映像コンテンツを共通に利用可能にする事が考えられる．

2.2 映像形式の検討

前述の通り，自然画像も人工画像 (CG) も，ファイルフォーマットは異なっても，PC 系デバイスのディスプレイへの表示段階では RGB 値で表現されている．ディスプレイ上の RGB 値画素配列として扱えば，両者は同じデータとみなすことができる．また，PC デバ

イス上では，このような RGB 値の配列としての映像表現は一般的に用いられている．静止画では BMP 形式などがそうであり，デスクトップ画面の表示もそうである．しかし，データ量が莫大なため，蓄積系，ネットワーク経由での映像配信共に積極的に用いられる例はほとんどなかった．

しかし，帯域の拡大，ストレージ容量の増大により，これらを利用する事は現実的なアプローチとなりつつある．そこで，本研究ではファイルフォーマットとして，ディスプレイ上の RGB 値画素配列を用いる．また，PC 系デバイスのディスプレイ表示と同じ規格の採用により，ディスプレイ表示との親和性を高める事にも繋がる．

以上の点を考慮した結果，本研究で採用する映像方式の要求条件は以下の通りとなる．

1. RGB 各 8bit 計 24bit フルカラーの色表現
2. xGA 規格の画面サイズ，アスペクト比 4:3
   → PC 系デバイスへの親和性のため
3. 非圧縮，もしくは簡略な圧縮処理
   → 今回は処理系の単純化のため非圧縮

2.2.1　帯域要求

次章の実験では，本研究方式の画像転送系への利用を実装し，その性能評価と問題点の整理を行う．その前段階として，帯域要求の見積りを作成しておく．なお，今回は達成目標とするフレームレートを 60fps 以上と設定し，表 2.1 に，PC 系デバイスの表示規格の要求条件に基づいて，必要とされる帯域幅を示す．

表 2.1　解像度と要求される帯域幅

表示形式	解像度 (pixel)	データ量 (bit)	要求 fps	必要帯域 (Mbps)
QVGA	76800	1843200	60	105
VGA	307200	7372800	60	421
SVGA	480000	11520000	60	659
XGA	786432	18874368	60	1080

第 3 章

画像転送実験

本章では，前章において示した要求条件に従い，非圧縮 RGB ベースの映像転送系を構築し，映像転送実験及び検証を行う．作業は，画像の送信および受信プログラムの作成，Gigabit Ethernet(IEEE802.3ab) [6] を用いたネットワーク環境の構築，転送実験の実施という手順に沿って行った．

3.1　実験用プログラムの開発

画像転送実験を行うため，画像の生成と送信を行うプログラム，画像データの受信と表示を行うプログラムが必要となる．以下には，本研究で用いるプログラムが必要とする手続きの検討と，必要な機能の導出，その実装の過程を示す．

3.1.1　画像転送処理手続きの検討

画像送信側アプリケーションは，大きく分けて次の処理を行う必要がある．

1. 前処理

   グラフィックイメージ生成準備，Windows Socket の初期化，受信側アプリケーションへの接続要求，コネクションの確立．

2. 画像生成処理

   Windows GDI 関数を用い，メインメモリ上にグラフィックイメージの生成．

3. 画像データ転送処理

   グラフィックイメージを送信バッファにコピーし，受信アプリケーションへ送信．

3.1　実験用プログラムの開発

受信側アプリケーションでは

1. 前処理

   受信画像表示用ウィンドウの初期化，画像データ受信用スレッド起動，サーバソケット準備，受信待機.

2. 画像データ受信処理

   送信側プログラムから転送された画像データを受信し，逐次表示用バッファにコピー.

3. 画像表示処理

   一画面分の画像データ受信完了後，表示用バッファの内容を，ウィンドウの表示領域に描画.

といった処理が必要である．この両者の画像転送処理の流れを，図 3.1 に示す.

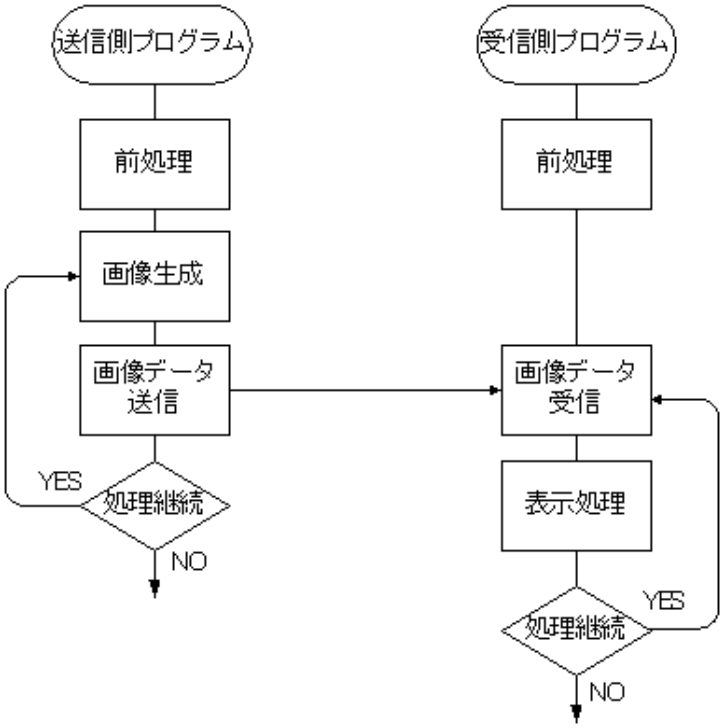

図 3.1　画像データ送受信の手順

これらの基本的な処理を実装し，必要なコンポーネントの組み合わせとしてアプリケーションを作成する．開発作業は，表 3.1 に示す環境で行った．また，プログラミング作業に

あたり，書籍 [7] [8] 及び WWW ページ [9] [10] を参考にしている．なお，送信側及び受信側プログラムのソースコードは，付録 A として文末に記載しておく．

表 3.1　開発環境

使用言語	C, Windows API
統合開発ツール	MS Visual C++ 6.0
	Borland C++ compiler
	Cpad for Borland C++ compiler

3.2　実験環境の構築

既存の環境において，送信者と受信者間を高速接続するため，1Gbit Ethernet を使用した．送信端末及び受信端末の 32bit PCI バスに 1000base-T Ethernet card を装着し，これを Enhanced CAT.5 クロスケーブルによって直接接続しネットワークを構成した．これは，超高速ネットワーク環境を実現するため，現状において得られる帯域により，高速性を確保するための措置である．今回の実験では，その他の通信ストリームの影響は考慮しないものとする．図 3.2 に実験環境を図示し，表 3.3 および表 3.2 に，実験に使用した端末の仕様を示す．

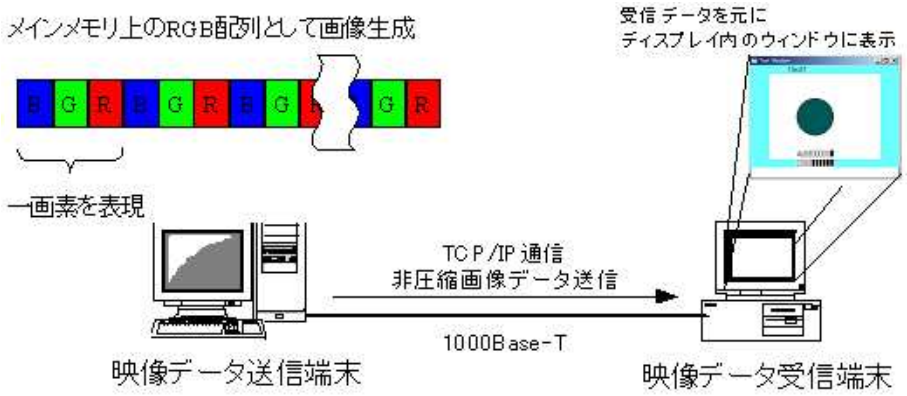

図 3.2　実験環境図

表 3.2　画像データ送信端末の仕様

CPU	Intel Pentium　600MHz
メモリ	PC100 SDRAM 128MB
チップセット	Intel 440BX
AGP	AGP 1.0 2x
PCI	32bit 33MHz
グラフィックスカード	Canopus SPECTRA Light 1200
NIC	CentreCOM LA1000-PCI-T
NIC ドライバ	LGTXND4.SYS ver.1.0.10723.0
OS	MS Windows NT Professional SP6a

表 3.3　画像データ受信端末の仕様

CPU	Intel Pentium 4 1.5GHz
メモリ	RDRAM 256MB
チップセット	Intel 850
AGP	AGP 2.0 4x
PCI	32bit 33MHz
グラフィックスカード	nVIDIA GeForce2 MX
NIC	CentreCOM LA1000-PCI-T
NIC ドライバ	LGTXND5.SYS ver.2.0.10.0
OS	MS Windows 2000 Professional SP2

3.3　画像転送実験の実施

画像転送実験として，以下の内容を実施した．

解像度を 320x240(QVGA) から 1024x768(XGA) まで変化させ，送信プログラムにおいて画像を生成，画像データを転送し，受信プログラムで送信されたデータを元に画像を生成，標準出力へ表示するという処理を 1 サイクルとする．このサイクルを連続 2,000 回繰り返すことで，2,000 フレーム分の動画転送処理を行った．

3.4　実験結果

結果の出力として，最初の一枚の画像転送開始から 2,000 枚目転送終了までにかかる時間 (秒) を計測し，一秒あたりの転送枚数 (fps) を求めた．また，受信データ量 (MByte) を元に平均転送速度 (Mbps) を求めている．表 3.4 に転送実験の結果を，図 3.3 に解像度とフレームレート，平均転送時間の関係を示す．

表 3.4　解像度別実験結果

解像度 (横 x 縦)	受信データ量 (MB)	転送時間 (秒)	fps	平均転送速度 (Mbps)
320x240(QVGA)	439.4	30.0	66.5	116.9
400x300	686.6	42.6	46.9	128.9
480x360	988.7	57.8	34.6	136.8
560x420	1345.8	85.8	23.2	125.3
640x480(VGA)	1757.8	108.5	18.4	129.4
800x600(SVGA)	2746.5	175.9	11.3	124.8
1024x768(XGA)	4500.0	316.4	6.3	113.7

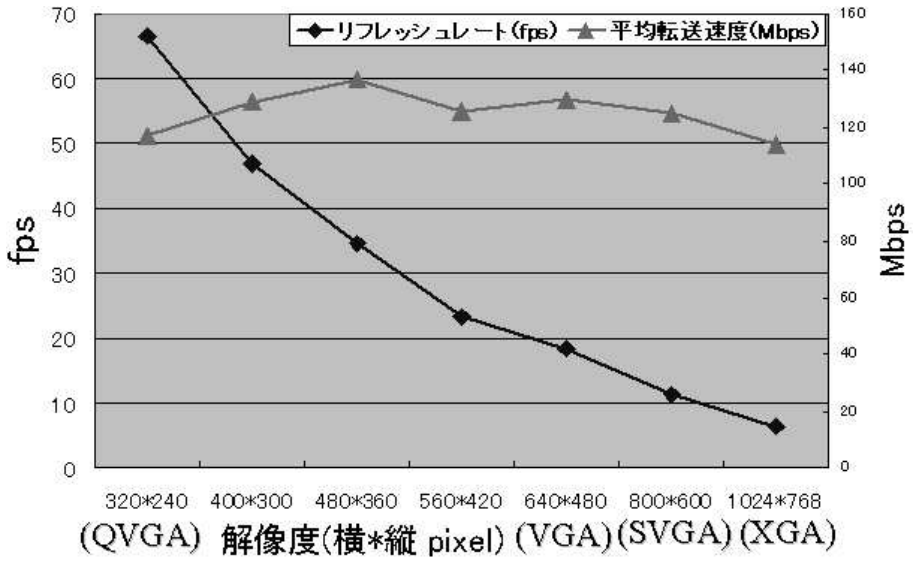

図 3.3 解像度と転送時間，平均転送速度の関係

3.5 考察

送信プログラムと受信プログラム双方で，送信および受信データ量の計測と比較を行い，データの損失が起こっていないことを確認した．これにより，通信路上のエラーによるフレームの欠落は起こっていないと判断する．

実験の結果より，QVGA 程度の解像度において，60fps 以上の表示速度が達成可能である事が示された．それ以上の解像度については，十分な表示速度の達成には至っていない．これは本研究で用いているファイルフォーマットに従った場合，転送される画像データ量が莫大であるためではあるが，Gigabit Ethernet の転送処理能力があれば，VGA サイズでも 60fps 以上の表示速度が得られるはずである．このことから，転送処理の中にボトルネックが存在している事が考えられる．以下には，そのボトルネックの原因と考えられるものを挙げ，その検証を行う．

3.5.1 描画処理遅延の影響

最初に考えられる原因として，送信及び受信アプリケーションにおける描画処理速度に原因があることが考えられる．そこで，実験に使用したアプリケーションから描画処理を行う

部分を取り除き，データ転送のみを行うようにしたバージョンを用いて，データ転送にかかる時間を計測する．

具体的には，送信アプリケーションから画像生成処理部をコメントアウトし，アプリケーション初期化時に生成される，一枚の同じ画像の転送を続けるようにする．受信アプリケーションからは，ウィンドウの再描写処理を行う関数をコメントアウトし，一画面分のデータを受信完了した後も，一切再描画処理を行わないようにした．

描画処理を省略した転送実験

描画処理に関する機能を全て停止したアプリケーションにより，データ転送実験をおこなった．今回の実験では，QVGA，VGA，SVGA，XGA の各解像度について転送実験を行った．次の表 3.5 及び図 3.4 にこの実験の結果を示す．

表 3.5　省略版実験結果

解像度 (横 x 縦)	受信データ量 (MB)	転送時間 (秒)	平均転送速度 (Mbps)
320x240(QVGA)	439.4	19.7	178.1
640x480(VGA)	1757.8	64.0	219.6
800x600(SVGA)	2746.5	103.7	211.6
1024x768(XGA)	4500.0	193.6	185.9

考察

実験結果より，概ね転送能力が 60〜70Mbps 以上向上している事が分かる．これにより，グラフィックデータを処理するための，端末の処理能力がボトルネックの一つとなっていたことが示された．しかし，それでもなお 220Mbps 未満の転送能力しか達成しておらず，その他のボトルネックの検討の余地がある．

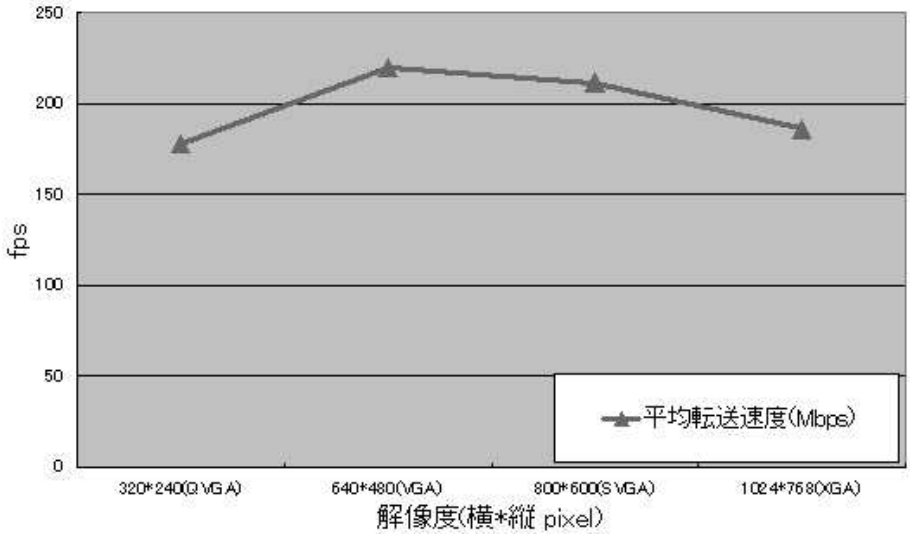

図 3.4　画像生成省略版，解像度と平均転送速度の関係

3.5.2　PCI バス幅

今回の実験で使用した端末は，どちらも 32bit，33MHz 動作の PCI バスに 1000base-T Ethernet card を接続している．この PCI バスの，理論上の最大通信速度は 1056Mbps である．そのため，PCI バスの帯域を全て NIC が占拠しなければ，ギガビット級の通信速度を達成することができない事が分かる．しかし，実際には PCI バスを，サウンドカードや，その他のマザーボード上のハードウェアと共有しているため，十分な帯域を確保できなかったという事が考えられる．

3.6　RGB ベースの圧縮法の検討

現在の環境下において RGB ベースの映像転送を行うためには，データ量削減策を講じる必要がある．そこで，以下では画像データのサンプルとして，標準画像データベースより SIDBA77 Girl(図 3.5) [11] の生データを用いて，RGB ベースの圧縮法の検討に取り組む．

3.6.1　検証作業

今回，以下の内容の検証作業を行った．

1. 標準画像本来のエントロピーの算出
2. 輝度成分自身と，輝度成分と各 RGB 成分との差分を利用した場合のエントロピーの算出

図 3.5　SIDBA 77 Girl

最初の項目は，何も操作を加えない状態で，その画像にどのぐらい圧縮をかけられるかを示し，評価基準とするための作業である．

次の項目は，RGB 各値すべてに強い影響を及ぼしている輝度成分の影響を調べておくための作業である．この方法は，一度 YUV 系の介在を許している点で RGB ベースの手法ではないが，輝度成分の影響力を知り，今後の参考とするために必要であると考え，その検証を行った．

3.6.2　検証結果

以下の表 3.6 に検証作業 1，表 3.7 に検証作業 2 の結果を示す．

表 3.6　raw データのエントロピー

信号の種類	R	G	B
エントロピー	6.42	6.44	6.38
必要符号長 (bit)	7	7	7

3.6.3　考察

raw データのエントロピーより，加工無しの元画像の符号化には，RGB 各 7bit，計 21bit のデータ長が必要なことが分かる．輝度成分との差分を用いた場合では，R-Y の符号化に

表 3.7　輝度成分と RGB との差分を用いた方式のエントロピー

信号の種類	R-Y	G-Y	B-Y
エントロピー	5.88	4.24	4.51
必要符号長 (bit)	6	5	5

6bit，G-Y に 5bit，B-Y に 5bit が必要で，この内符号長の短い G-Y と B-Y，そして Y 信号 8bit の組み合わせの計 18bit で符号化できることが分かる．

結果として，元データより加工後のほうが，1 画素あたり 3bit，約 15%データを削減できることが分かった．この数字は画像圧縮処理としては得られるものとしては小さなものであり，今後この方針とは異なるアプローチから圧縮処理の検討を行う必要がある．

第 4 章

結論

4.1 総括

本研究では，自然画像，CG を統一的に利用可能にすることを目指し，その一つの解決策として，RGB ベースの映像フォーマットの提案を行った．またその実装として，RGB ベースの画像転送系の構築を行い，画像転送実験と性能評価を行った．

実験の結果，QVGA 程度の解像度の画像を 60fps 以上で転送することを達成した．しかし，高解像度域においては十分な速度達成には至らなかった．そこで，処理のボトルネックの所在を明確にするため，画像描画処理を省略したアプリケーションを作成し，再び転送能力の評価を行った．結果として，60〜70Mbps 以上の転送能力向上が図られ，ここに一つのボトルネックが存在していることを示した．

さらに，現状の帯域制限の元で RGB ベースの映像転送系を実現するため，圧縮処理の検討を行った．ここでは，輝度成分と RGB 値との差分の利用では，15%前後しかデータ削減が期待できないことが示された．

4.2 今後の課題

現状において RGB ベースの映像利用を可能とするためには，RGB ベースの軽負荷圧縮処理が必要となる．その検討にあたり考慮する点として，

1. 10 分の 1 程度の圧縮率
2. RGB 値に基づいた圧縮処理

3. 走査線間の相関性の利用

といった点が挙げられる．

ここで，走査線間の相関性を，再び図 4.1 の標準画像 SIDBA 77 Girl(印刷上白黒画像であるが，実際は RGB24bit フルカラー画像) [11] を用いて以下に示す．

まず, 画像最上段の走査線 3 つ分の RGB 値配列を切り出し，Y 軸を輝度値として X 軸方向に直線に並べる．この手順の詳細は図 4.2 に示す．そうして作成した図 4.3 のグラフから見て取れるように，走査線の画素値が示す曲線は，隣接する走査線のものと非常に形状が似通っていることが分かる．この曲線の類似性を利用して，データから冗長性を取り除く圧縮法の検討を現在進めている．

図 4.1　SIDBA 77 Girl

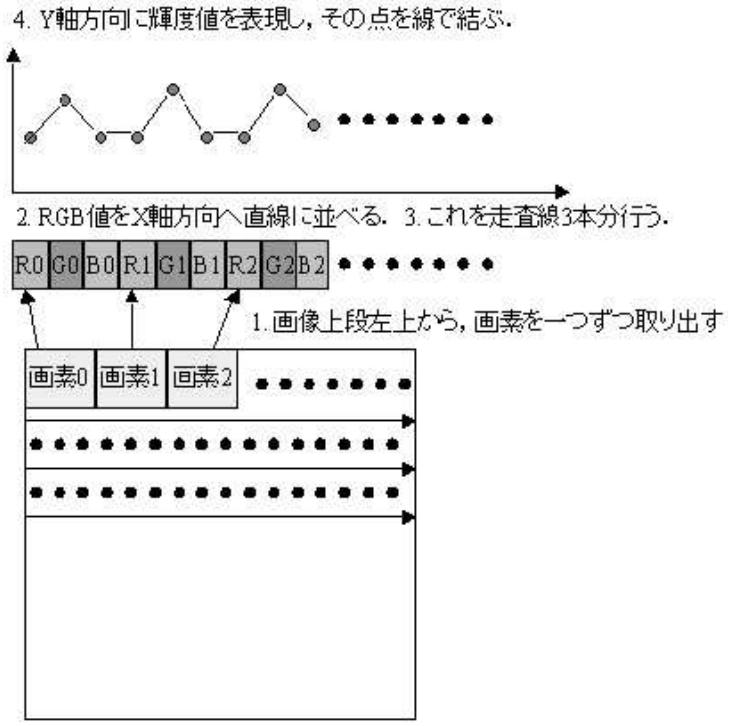

図 4.2　グラフ作成手順の詳細

4.2　今後の課題

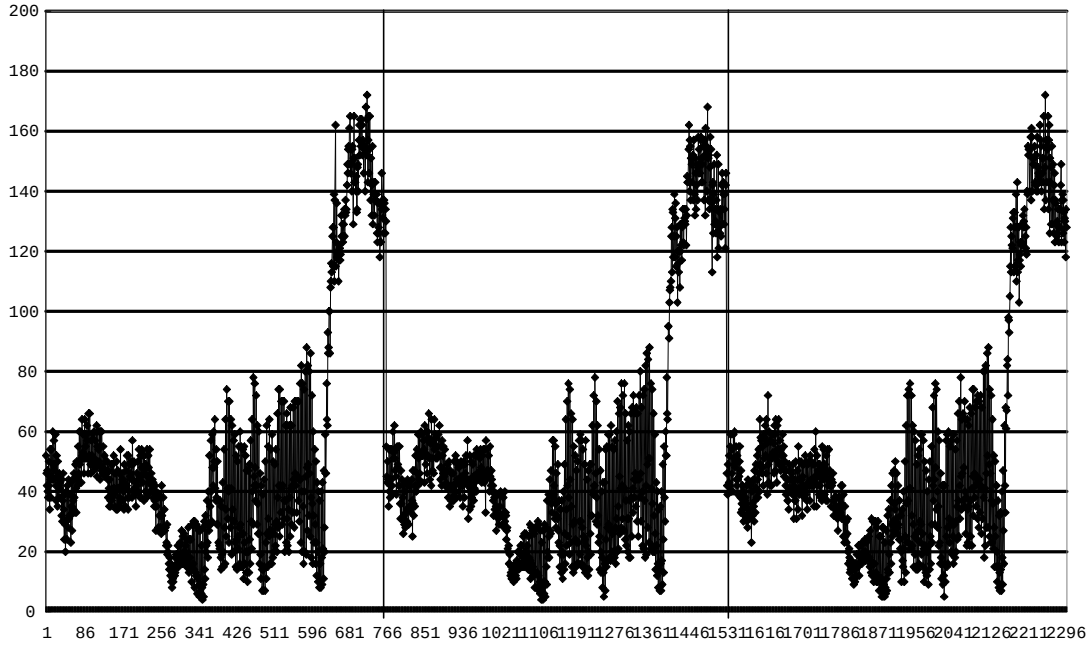

図 4.3　走査線間の RGB 値の関連性

謝辞

学年担当のアドバイザーとして，不出来であった私の更生に尽力して頂いたうえ，研究室の一員として，よき研究者となるべく学業への取り組みから生活態度に至るまで，様々なご指導を頂いた島村 和典教授に深く感謝します．また，副査を務めて頂いた岡田 守教授，竹田 史章教授にも深く感謝します．そして，島村研究室 mAV サブグループのリーダーとして，直接研究指導にあたり，多くの有益な情報，アドバイスを頂いた中平 拓司氏に深く感謝します．最後に，mAV サブグループ所属の修士院生で，様々なアイデアや意見を頂いた山岡 徹也氏，放送・通信機構 高知通信トラヒックリサーチセンターの加藤 寛治研究員，神田 敏克研究員，高松 希匠研究員に感謝します．

参考文献

[1] 2002 年 2 月 6 日時点 WWW ページ, “ 総務省 DSL 普及状況公開ページ ”,
URL http://www.soumu.go.jp/joho_tsusin/whatsnew/dsl/

[2] 2002 年 1 月 31 日時点 WWW ページ, “ 平成 13 年版 情報通信白書 ”,
URL http://www.soumu.go.jp/hakusyo/tsushin/index.html

[3] 小野文孝，渡辺裕 共著, “ 高度映像技術シリーズ 1 国際標準画像符号化の基礎技術 ”,
コロナ社，1998.

[4] 2002 年 1 月 31 日時点 WWW ページ, “ PIONEER R&D 技術解説 ”,
URL http://www.pioneer.co.jp/crdl/tech/index.html

[5] 2002 年 1 月 31 日時点 WWW ページ, “ The Programmer’s File Format Collection ”,
URL http://www.wotsit.org/

[6] 羽鳥光俊 監修, “ ポイント図解式 コンピュータ通信放送 標準事典 ”，アスキー，1999.

[7] 山本信雄 著, “ プログラミング学習シリーズ Visual C++ ”，翔泳社，2000.

[8] 田口景介 著, “ 実習 C 言語 新版 ”，アスキー，1999.

[9] 2001 年 10 月 8 日時点 WWW ページ, “ Windows プログラミング研究室 ”,
URL http://www.sm.rim.or.jp/ shishido/windows.html

[10] 2001 年 10 月 8 日時点 WWW ページ, “ Hey! Java Programming! C 言語ソケット ”,
URL http://www.mars.dti.ne.jp/ torao/program/socket/

[11] 2002 年 2 月 6 日時点 WWW ページ, “ 画像アーカイブのホームページ ”,
URL http://www.tlab.te.noda.sut.ac.jp/Image-Archive.html

付録 A

プログラムソースコード

A.1 画像受信側プログラム

```
1   #include <windows.h>
2   #include <winsock.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6
7   /* 解像度設定 */
8   #define WIDTH 1024
9   #define HEIGHT 768
10  /* 転送一回当りのライン数決定 */
11  #define MLT 2
12
13  LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
14
15  BITMAPINFO BmInfo;
16  BITMAPINFOHEADER BmInfoHed;
17  LPBYTE lpBMP;
18  HANDLE thread1;
19  int count=0,total=0;
20
21  DWORD WINAPI rgbv(LPVOID lpv) {
22
23          int i,j,k;
24          /* 通信に関する諸変数 */
25          WSADATA info;                   /* Winsock 情報用構造体 */
26          SOCKET serv, newserv;           /* 通信に使用するソケット */
27          struct sockaddr_in addr;        /* アドレス情報構造体 */
28          u_long inaddr;                  /* ホストの IP アドレス */
29          u_char msgs[] = "1\0";
30          u_char errm[] = "2\0";
31          int addrs;
32          int res;
33
34          /* 画像表示に関する諸変数 */
35          HWND hwnd = lpv;
36          LPBYTE newlpBMP;
```

```
37          BYTE *bm;
38          int hline,wline,bmsize;
39
40          hline  = (int)(HEIGHT / MLT);
41          wline  = (int)(WIDTH * MLT);
42          bmsize = (int)(wline * 3);
43
44          bm     = (BYTE *)malloc(bmsize+2);
45
46          for(k=0;k<bmsize+2;k++){
47                  bm[k]='\0';
48          }
49
50          newlpBMP=(LPBYTE)malloc(WIDTH*HEIGHT*3);
51
52
53          /* Winsock の初期化（バージョン 2.0 を要求） */
54          WSAStartup(0x0002, &info);
55
56          inaddr = inet_addr("172.21.33.120");
57
58          /* アドレスの設定 */
59          addr.sin_family = AF_INET;      /* インターネットアドレス */
60          addr.sin_port   = htons(1100);  /* ポート番号設定 */
61          addr.sin_addr   = *(struct in_addr*)&inaddr;/* アドレス設定 */
62          memset(addr.sin_zero, 0x00, 8);             /* パディング */
63
64          /* ストリームソケットの構築 */
65          serv = socket(AF_INET, SOCK_STREAM, 0);
66
67          /* バインド */
68          bind(serv, (struct sockaddr*)&addr, sizeof(addr));
69
70          /* サーバソケットの設定 */
71          listen(serv, 2);
72
73          addrs=sizeof(addr);
74          /* 接続受付 */
75          newserv = accept(serv, (struct sockaddr*)&addr, &addrs);
76
77          while(1) {
78                  for (i=0;i<hline;i++) {
79                          res=recv(newserv, bm, bmsize, 0);
80                          if(res <= 0) {break;}
81
82                          total=total+res;
83
84                          for (j=0;j<bmsize;j++) {
85                                  newlpBMP[i*bmsize+j]=bm[j];
86                          }
87                          if(send(newserv, msgs, strlen(msgs), 0)<=0){
```

```
88                                      break;
89                              }
90                      }
91                      if(i < (hline-1)){break;}
92                      lpBMP=newlpBMP;
93                      InvalidateRect(hwnd,NULL,FALSE);
94
95                      count++;
96              }
97
98              free(bm);
99              /* ソケットの破棄 */
100             closesocket(newserv);
101             closesocket(serv);
102             /* Winsock 終了処理 */
103             WSACleanup();
104             return 0;
105     }
106
107     int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
108                         PSTR szCmdLine, int iCmdShow){
109
110             HWND hwnd;
111             MSG  msg;
112             WNDCLASSEX  wndclass ;
113
114             wndclass.cbSize        = sizeof(wndclass);
115             wndclass.style         = CS_HREDRAW | CS_VREDRAW;
116             wndclass.lpfnWndProc   = WndProc;
117             wndclass.cbClsExtra    = 0;
118             wndclass.cbWndExtra    = 0;
119             wndclass.hInstance     = hInstance;
120             wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION);
121             wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW);
122             wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
123             wndclass.lpszMenuName  = NULL;
124             wndclass.lpszClassName = "Test Window";
125             wndclass.hIconSm       = LoadIcon (NULL, IDI_APPLICATION);
126
127             RegisterClassEx (&wndclass);
128
129             hwnd = CreateWindow ("Test Window",
130                                  "Test Window",
131                                  WS_OVERLAPPEDWINDOW,
132                                  0,0,
133                                  WIDTH+8,HEIGHT+26,
134                                  NULL,
135                                  NULL,
136                                  hInstance,
137                                  NULL);
138
```

```
139             ShowWindow(hwnd,iCmdShow);      /* ウインドウを表示 */
140
141             BmInfoHed.biSize=sizeof(BITMAPINFOHEADER); /* 構造体の大きさ */
142             BmInfoHed.biWidth=WIDTH;                   /* 横幅 */
143             BmInfoHed.biHeight=HEIGHT;                 /* 高さ */
144             BmInfoHed.biPlanes=1;                      /* プレーンの数 */
145             BmInfoHed.biBitCount=24;                   /* プレーンの色数 */
146             BmInfoHed.biCompression=BI_RGB;
147             BmInfoHed.biSizeImage=0;
148             BmInfoHed.biXPelsPerMeter=0;
149             BmInfoHed.biYPelsPerMeter=0;
150             BmInfoHed.biClrUsed=0;
151             BmInfoHed.biClrImportant=0;
152
153             BmInfo.bmiHeader=BmInfoHed;
154
155             /* ビットマップ用バッファ */
156             lpBMP = (LPBYTE)malloc(WIDTH*HEIGHT*3);
157
158             UpdateWindow(hwnd);
159
160             while (GetMessage (&msg,NULL,0,0)) { /* メッセージループ */
161                     TranslateMessage(&msg);
162                     DispatchMessage(&msg);
163             }
164             return msg.wParam;
165     }
166
167     LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
168                               LPARAM lParam){
169
170             HDC hdc;
171             int d1;
172             PAINTSTRUCT ps;
173             char strings[10];
174             char countr[10];
175
176             switch (iMsg) {
177
178                     case WM_DESTROY:
179                             /* ビットマップのバッファを解放 */
180                             free(lpBMP);
181                             PostQuitMessage(0);
182                             return 0;
183                             break;
184
185                     case WM_PAINT:
186
187                             hdc=BeginPaint(hwnd,&ps);
188                             /* ウィンドへ DIB の内容を表示 */
189                             StretchDIBits(hdc,0,0,WIDTH,HEIGHT,
```

```
190                                     0,0,WIDTH,HEIGHT,lpBMP,&BmInfo,
191                                     DIB_RGB_COLORS,SRCCOPY);
192
193                             itoa(total,strings,10);
194                             TextOut(hdc, 100, 180, strings, 10);
195
196                             itoa(count,countr,10);
197                             TextOut(hdc, 100, 200, countr, 10);
198
199                             EndPaint(hwnd,&ps);
200                             break;
201
202                     case WM_RBUTTONUP:
203                             thread1=CreateThread(NULL,0,rgbv,hwnd,0,&d1);
204                             SetThreadPriority(thread1,
205                                                     THREAD_PRIORITY_HIGHEST);
206                             break;
207             }
208             return DefWindowProc (hwnd, iMsg, wParam, lParam);
209     }
```

A.2　画像送信側プログラム

```
1   #include <windows.h>
2   #include <winsock.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   #include <time.h>
7
8   /* 解像度設定 */
9   #define WIDTH 1024
10  #define HEIGHT 768
11  /* 転送一回当りのライン数決定 */
12  #define MLT 2
13
14  void error(const char* msg);
15
16  int main(void){
17          clock_t time;
18          WSADATA info;                   /* Winsock 情報用構造体    */
19          struct sockaddr_in addr;        /* ホストの IP アドレス    */
20          u_long inaddr;                  /* インターネットアドレス */
21          SOCKET sender;                  /* 通信用ソケット          */
22          int  i,j,k,x,y,res;
23          BYTE *bm;
24          char tt[2];
25          int sendata=0,recvdata=0;
26
```

```
27          /* 画面操作部 */
28          HDC hdc,hdcMem;
29          HBITMAP hBMP;
30          LPBITMAPINFO lpDIB;
31          LPBYTE lpBMP;
32
33          int hline,wline,bmsize;
34          COLORREF color,bkcolor;
35          /* 描画用ブラシ */
36          HBRUSH brush,oldbrush;
37
38          hdc=CreateDC("DISPLAY",NULL,NULL,NULL);
39
40          hline=(int)(HEIGHT / MLT);
41          wline=(int)(WIDTH * MLT);
42          bmsize=(int)(wline*3);
43
44          /* DIB用メモリ確保 */
45          lpDIB=(LPBITMAPINFO)malloc(sizeof(BITMAPINFO));
46          /* 送信バッファ用メモリ確保 */
47          bm=(BYTE *)malloc(wline*3);
48
49          /* DIBSection 用BITMAPINFO構造体設定 */
50          lpDIB->bmiHeader.biSize=sizeof(BITMAPINFOHEADER);
51          lpDIB->bmiHeader.biWidth=WIDTH;
52          lpDIB->bmiHeader.biHeight=HEIGHT;
53          lpDIB->bmiHeader.biPlanes=1;
54          lpDIB->bmiHeader.biBitCount=24;
55          lpDIB->bmiHeader.biCompression=BI_RGB;
56          lpDIB->bmiHeader.biSizeImage=0;
57          lpDIB->bmiHeader.biXPelsPerMeter=0;
58          lpDIB->bmiHeader.biYPelsPerMeter=0;
59          lpDIB->bmiHeader.biClrUsed=0;
60          lpDIB->bmiHeader.biClrImportant=0;
61
62          /* DIB と画面のDC からDIBSection を作成 */
63          hBMP=CreateDIBSection(hdc,lpDIB,
64                                DIB_RGB_COLORS,&lpBMP,NULL,0);
65
66          hdcMem=CreateCompatibleDC(hdc); /* メモリDC を作成 */
67          SelectObject(hdcMem,hBMP); /* メモリDC にDIBSection を選択 */
68
69          /* Winsock の初期化（バージョン 2.0 を要求） */
70          res = WSAStartup(0x0002, &info);
71          if(res == SOCKET_ERROR) error("Winsock initialize.");
72
73          inaddr = inet_addr("172.21.33.120");
74
75          /* アドレスの設定 */
76          addr.sin_family = AF_INET;    /* インターネットアドレス */
77          addr.sin_port = htons(1100);  /* ポート（変換して代入） */
```

```
78          addr.sin_addr = *(struct in_addr*)&inaddr; /* アドレス設定 */
79          memset(addr.sin_zero, 0x00, 8);            /* パディング   */
80
81          /* ストリームソケットの構築 */
82          sender = socket(AF_INET, SOCK_STREAM, 0);
83          if(sender == INVALID_SOCKET) error("Socket creation.");
84
85          /* ホストと接続 */
86          res = connect(sender, (struct sockaddr*)&addr, sizeof(addr));
87          if(res == SOCKET_ERROR) error("Connection.");
88
89          /* 初期描画処理 */
90          x=0;
91          y=0;
92          bkcolor=RGB(100, 255, 255);
93          SetBkColor(hdcMem, bkcolor);
94          Rectangle(hdcMem,0,0,WIDTH,HEIGHT);
95          color=RGB(0,255,0);
96          brush=CreateSolidBrush(color);
97          oldbrush=SelectObject(hdcMem, brush);
98          Ellipse(hdcMem, 100,70,180,150);
99
100         /* 時間計測開始 */
101         time = clock();
102
103         for (k=0;k<2000;k++) {
104                 /* 画像の生成 */
105                 if(x<280 && y == 0){
106                         x++;
107                 } else if(x==280 && y<200){
108                         y++;
109                 } else if(x>0 && y==200) {
110                         x--;
111                 } else {
112                         y--;
113                 }
114                 TextOut(hdcMem, x, y, "1test1", 6);
115                 color=RGB(y,min(255,x),max(0,min(255,(x-y))));
116                 brush=CreateSolidBrush(color);
117                 SelectObject(hdcMem, brush);
118                 Ellipse(hdcMem, 100,70,180,150);
119
120                 /* ビットマップ送信処理 */
121                 for (i=0;i<hline;i++) {
122                         for(j=0;j<wline;j++) {
123                                 /* ピクセルの RGB 成分を取得 */
124                                 bm[j*3+2]=lpBMP[j*3+2+i*bmsize];
125                                 bm[j*3+1]=lpBMP[j*3+1+i*bmsize];
126                                 bm[j*3]  =lpBMP[j*3+i*bmsize];
127                         }
128                 /* 画像データ送信 */
```

```
129                     res = send(sender, bm, bmsize, 0);
130                             if(res == SOCKET_ERROR){
131                                     error("Data Sending.");
132                             }
133                     sendata=sendata+res;
134
135                     /* 受信完了信号の受信 */
136                     res = recv(sender, tt, sizeof(tt), 0);
137                             if(res == SOCKET_ERROR){
138                                     error("Data Reading.");
139                             }
140                     recvdata=recvdata+res;
141                     }
142             }
143
144             /* 終了時間計測 */
145             time = clock() - time;
146             scanf("%s", &tt);
147
148             /* ソケットの破棄 */
149             res = closesocket(sender);
150             if(res == SOCKET_ERROR) error("In closing socket.");
151
152             /* Winsock 終了処理 */
153             res = WSACleanup();
154             if(res == SOCKET_ERROR) error("Cleanup.");
155
156             /* ブラシを元に戻す */
157             SelectObject(hdcMem, oldbrush);
158
159             /* デバイスコンテキスト開放 */
160             DeleteDC(hdc);
161             DeleteDC(hdcMem);
162
163             /* オブジェクト削除 */
164             DeleteObject(hBMP);
165             DeleteObject(brush);
166             DeleteObject(oldbrush);
167
168             /* メモリ開放 */
169             free(lpDIB);
170             free(bm);
171
172             /* 結果表示 */
173             printf("%.0f ms \n", (double)((time*1000)/CLOCKS_PER_SEC));
174             printf("sent data %d\n", sendata);
175             printf("recieved data %d\n", recvdata);
176             scanf("%s", &tt);
177             return EXIT_SUCCESS;
178     }
179
```

```
180   /* エラー処理 */
181   void error(const char* msg){
182           printf("Winsock Error[%d]: %s\n", h_errno, msg);
183           exit(EXIT_FAILURE);
184           return;
185   }
```