# Theoretical Science Group

## 理論科学グループ

# 部報 207 号
# — 新入生歓迎号 —

# 目　次

# 熱烈歓迎！新入生！

## 名前一覧

1997/05/07 現在編集長が把握している新 1 年生です。

| | | | | | |
|---|---|---|---|---|---|
| 730156B | 石橋 しのぶ | 730337I | 山室 和澄 | 730431D | 奥野 緑 |
| 740024C | 菅 哲朗 | 740029H | 小林 浩士 | 740048G | 野口 博史 |
| 740054F | 松尾 優輝 | 740060E | 柳瀬 健吾 | 740061H | 山岸 史典 |
| 740084I | 工藤 誠 | 740117D | 村上 恭基 | 740184B | 和井田 寛則 |
| 740208I | 今野 俊一 | 740271E | 鈴木 真介 | 740311B | 井上 悠介 |
| 740340B | 辻河 亨 | 740374G | 岡崎 直観 | 740441H | 河野 匡彦 |
| 740443D | 齊藤 文彦 | 740466E | 森田 純一郎 | 740543G | 塚田 与志也 |
| 740581E | 坂尾 要祐 | 740688I | 増田 太郎 | 740746B | 井野 雄介 |
| 740770G | 内藤 誠一郎 | 740773F | 登尾 大地 | 740811G | 高松 尚司 |
| 740833E | 安達 雅直 | 740839C | 大久保 亮 | 740851A | 柴田 剛志 |
| 740895G | 佐藤 大輔 | 741048H | 長谷川 利晴 | 741094A | 中 暁洋 |
| 741119A | 伊藤 俊輔 | 741130F | 木野 泰之 | 741133E | 佐伯 修祐 |
| 741151A | 福林 一平 | 741203E | 本多 祐樹 | 750317C | 高野 聡子 |
| 750491F | 森田 雅樹 | 641014B | 斉藤 晋一郎 | | 小玉 浩平 |

1997 年度入学駒場教養学部生のメールアドレスは ecc の慣例に従い、学生証番号から

g7xxxxx@komaba.ecc.u-tokyo.ac.jp

となります。

# 自己紹介文章

この自己紹介は、主に情報処理棟分科会第 1 回目の課題で提出した人がほとんどという前提を踏まえた上でお読みください。

編集長：すーゆー

## 菅 哲朗

こんにちは、長崎県出身の菅といいます。現在ひとり暮らしの毎日です。コンピュータの経験は、ゼロなのですが、大学に入り ID ももらったことだし、コンピュータに触れてみたいと思い、入会を希望しました。洗濯機がないので、洗濯は結構きついです。自炊をするにも、コンロはひとつでながしは狭くいかんともしがたい状況です。

## 小林 浩士

ハンドル：KOBA

秋田から来ました。趣味は音楽鑑賞とプロ野球観戦で、ストーン・ローゼズと桑田投手が好きです。コンピューターは使ったことがありません。暇なひとは教えて下さい。

## 野口 博史

新入生の野口です。自己紹介を書いておきます。

高校の時は 98 を持っていたので GAME をよくしていて、それなりに 98 のことや、DOS や BASIC は知っているのですが、C 言語などは全く知りません。大学にはいったら勉強しようとおもっていたので、この TSG にはいりました。これからよろしくおねがいします。

## 工藤 誠

理科一類 5 組の工藤 誠です。最近忙しかったので部室にはあまり顔をだしていません。コンピューターの勉強をしたいとおもってます。コンピューターにはあまり詳しくないので、みなさん色々教えてください。

## 村上 恭基

大学に入るまでパソコンを 触ったことがない僕が、この由緒正しき TSG に入部したことはある意味無謀だったかもしれません。しかし、やる気だけはあると思います。皆様には御迷惑をおかけするかもしれませんが、最後まで見捨てないでやってください。よろしくお願いします。

## 井上 悠介

こんにちわ。ただいま課題を申し渡された井上と申します。一応自己紹介をしろとのことで。

フルネームは井上悠介、福岡県の北九州市門司区出身です。僕の家からはあの関門橋が見えます。高校は、知る人は知っている、久留米大学附設高校という学校です。附設なんて知らないけど附属なら聞いたことがあるとかいう人もいますが (新聞に載った時も久留米大学附属高校となったことがある) 本当は附設高校です。この TSG には知っている人がいないようで、ちょっと淋しいと思っていますがしょうがないんでしょう。

それでは。

## 辻河 亨

理 I・10 組の辻河といいます。よろしくお願いします m(__)m

所有マシンが Libretto だけ (他にはお嫁に貸し出している ThinkPad 220 だけ) という状況なので、何をやるにも制約がついて不便ですね。例えば今日などは、カードスロットに SCP-55A を差したまま (これは発熱がすさまじい)12 時間の連続駆動試験をやっていたところ、HDD から異音が発生 (T_T) やむなく冷蔵庫で緊急冷却処分となり、その後は氷で冷やしながら使うはめになりました、まる。

## 岡崎 直観

**名前**　岡崎 直観 [okazaki naoaki]
**所属**　理科 I 類 211 組
**出身高校**　宇都宮高校 (栃木県)
**使用機種**　FMV-DESKPOER TE　2020MB HDD / 48MB MEM / DB-XG50 MIDI
**パソコン歴**　7 年 1 年 (BASIC) / 1 年 (MASM) / 2 年 (QuickC) / 残り (その他) }
**専門分野**　つい最近までは、PC98 の MS-DOS のゲーム・プログラミング (RPG の制作) をしていましたが、見切りをつけ、現在、Borland C++Builder で、WINDOWS プログラムを勉強しながらマップエディタを制作中です。あと 2ヶ月後には多分、DirectX のプログラムをしていると思います。
**趣味**　ゲームに関しては、ぷよぷよしかできません。一人暮らしをしていて、対戦する機会が減っているので、ぷよぷよが好きな方は勝負をしましょう。また、音楽が好きで、音楽系のサークルに加入しています。高校のときは部活でバドミントンをやっていた関係で、バドミントンのサークルにも加入しようと思っています。
**おまけ**　現在、大学のＥＣＣにメールがほとんど届きません。暇な人は出してください。

## 齊藤 文彦

```
<HTML>
<BODY BGCOLOR="#AAAAFF">
<H1><CENTER>自己紹介</CENTER></H1><BR>
<TABLE>
<TR><TD>名前</TD><TD>齊藤文彦</TD></TR>
<TR><TD>学年・科類</TD><TD>理科 1 類 1 年</TD></TR>
<TR><TD>住所</TD><TD>東京都杉並区成田西１－８－７</TD></TR>
<TR><TD>野望</TD><TD>Java 分科会を作る</TD></TR>
</TABLE>
```

| | |
|---|---|
| ハンドル | Fum |
| 所属 | 理科一類 12 組 (ドイツ語) |
| 出身高校 | 桐朋高校 |
| メールアドレス | VFD06645@niftyserve.or.jp |
| | g740443@komaba.ecc.u-tokyo.ac.jp |
| 使用機種 | IBM Aptiva J3V |
| | EPSON PC-486SR |

パソコンは中 1 からやってます (最初のマシンは PC-9801DX)。専門はプログラミングで、BASIC、Quick BASIC、Turbo C などを使ってました。86 アセンブラ (MASM) も少しかじったことがあります。でもブランクが大きい (かといって勉強してたわけではない) ので、C の関数とか結構忘れてます (^^;)。

最近はネットワークにこってて、父の会社のイントラネット (NT Server+IIS+Access97) をつくったりもしてます。

HTML は一通り書けるようになりました。DOS&Mac 歴は 6 年ですが、Windows はこの 4 月に使い始めたばかりです (なのになぜかいきなり NT Server)。

ゲームは嫌いじゃないけど殆どやりません。だからとってもヘタです。

現在 Java 使いになるべく、Visual Cafe を買って勉強中です。

## 森田 純一郎

理科一類 12 組

いやーなんなんでしょうかねー。どうしてパソコン関係の本はあそこまで難しいのでしょうか。と、いうわけで現在かなり苦しんでいます。こんな自分ですが、今後ともよろしくお願いいたします。もし苦しんでいるところを見たら助けてやって下さい。思いつくまま書いていたら自己紹介という当初の目的からかなりそれてしまったのでどうしましょうか。すこし困ってます。

しかたがないので自分の故郷の事でも書こうと思います。自分は佐賀県から来ました。だからなんなんだとつっこまれたらつらいです。よく考えたら何もないです。せっかくだから佐賀が九州にあるということだけでも覚えておいてください。

## 坂尾 要祐

**所属** 理科 1 類 15 組

**趣味** ゲーム、ぶらぶらすること

**弱点** 痛いのが苦手、ホラーもすごく苦手 (特に不意打ち的なものに弱い)

**目標**

1. プログラミングができるようになれるぐらいにコンピュータをうまく使えるようになる。
2. PS のゲームに慣れる。
3. シューティングゲームの腕をあげる。(初プレイ時に 1 コインで 5 面ぐらいまでいけるようになる)

**要望** 部室に PS のゲームの ”AZITO” を入れて欲しい (笑)。

**コメント**　僕には少々トロいところがあって、みなさんの足をひっぱることもあるかもしれませんが、仲良くして下さいね。

**付録**　最近欲しいと思ってるサターンソフトのランキング (5/1 現在)

1. ティンクルスタースプライツ
2. わくわく 7
3. D&D コレクション
4. 重装機兵レイノス 2
5. メタルスラッグ
6. 蒼窮グレン隊 (グレンの漢字がでないー (泣))
7. ルームメイト
8. 機動戦士ガンダム外伝
9. BATSUGUN

(注:本人も書いてから恥ずかしくなったようなので、つっこまないで下さい)

## 増田 太郎

こんにちは 増田 太郎 です。

沖縄県出身で、あの有名な普天間基地の近くに住んでいました。(この事は (P ～[1]) に知られたくない？)

趣味は映画で「アラビアのロレンス」「スターウォーズ」「グーニーズ」「マイフェアレディ」「E・T」「バックトゥザフュチャー」「ポセイドン・アドベンチャー」 ...

このまま紙面をすべて産めると別のサークルに回されそうなのでやめておきますが、パソコンとの関係で書くと最初にパソコンかっこいいと思ったのは「ウォーゲーム」(見ていない人は是非一度見てください！)、最近では「インディペンデンスデー」(面白かったけど無視して見なくて可)。

とにかく、大学へ入れたらパソコンをしたいとずっと思っていました (よって完璧な初心者)。目標はいつかリストラにあった時、自分の腕だけでハリウッドに就職できること (もちろん CG 関係)。

というわけで、初心者の分際で TSG の一員になりましたが、とにかくパソコンで何かをしたいのでよろしくお願いします。

[1]編注:検閲削除 (笑)

## 井野 雄介

今日は始めて自分の手で netscape に行くことができたので、非常に良かったです。今自分が一番不安なのは、皆さんがやっているように「キーボードを見ないで、且素早く叩けるようになるか」ということです。普通の勉強に力をいれ、且シケ長をやっている私は、どうしても「キーボード」の叩き方の練習に費やせる時間が他の人より少なくなりがちです。そこで質問、キーボードを早く且見ないで叩けるようになるにはどのくらい時間がかかるでしょうか？また、やはり特別な訓練が必要なのでしょうか？その辺を詳しく解説して下さい[2]。

## 内藤 誠一郎

理科一類 1 年 19 組の内藤誠一郎です。少しでもシステムを使えるようになりたくて入りました。既成のソフトしか使ったことがなく、極めて初心者です。とりあえず情報教育棟を満足に利用できるレベルになり、もし余裕があるなら難しいことにも手を出してみたいと思います。

星が好きで地文研天文部に所属しているので、昼休みや休講時には大概学館 307A の部室にいると思います。

よろしくお願いします。

## 登尾 大地

理一 19 組 (坂本さんの下クラです) の登尾大地 (のぼりお だいち) です。香川県立高松高校出身で高校のときは吹奏楽でホルンをやってました。パソコンに関しては全くの初心者ですが、やる気は一応あるつもりなのでどうかよろしくお願いします。(非常に短くてすみません[3])

## 安達 雅直

安達です。

パソコンはこれまでほとんどさわったことはなかったんですけどなんとなくやってみたいと思って TSG にはいりました。

[2]すまん。こんな所に質問があったとは (笑)。タイピングだけど、ブラインドタッチやら標準位置やら偉そうなことを言う人は多いけど、結局自分の打ちやすい様に打つのが一番速いね。わざわざキータイプのためだけに練習をするというのは余りお薦めできることではないです。(あくまで編集者私見)

[3]編注:かまへん、かまへん (笑)

よろしくおねがいします。

## 大久保 亮

学籍番号は、740839C で、名前は 大久保 亮 といいます。

県立浦和高校出身で、家は埼玉県浦和市の駒場競技場 (浦和レッズの本拠地) の近くです (通学に 1 時間半ほどかかります)。

中学の時はバスケ部で、高校の時は物理部でした。高 1 の時は、アセンブラを勉強し、ポリゴンで学校を再現して、学校の中を動き回れるソフトを作りました。高 2 では、デジタル回路を少し勉強し、マイクでひろった音を増幅し、ノートパソコンにシリアル転送する回路を作りました。高 3 で、人間が話した声を聞きとるソフト (連続単語音声認識) や、オンライン手書き入力漢字認識ソフト、ニューロコンピュータをつかった学習などをやりました (Delphi を使用)。ただ、こう書くとすごそうに見えますが、ほとんど名ばかりのソフトです。

どれも、古いパソコンでやっていたので、最近のパソコンやソフト (Windows95 や Visual C++、Internet など) はまったく分かりません。TSG では、それらのことを学んだりしたいです。あと、3D や数値解析などに興味があります。

よろしくお願いします。

## 長谷川 利晴

長谷川 利晴です。

コンピュータは、高校の授業で、2 年くらい、ワープロのようなことをやったくらいです。TV ゲームをやっていて、自分もこういったものをつくってみたいとか思いました。使いこなせるようになれば、かっこいいかなとも思っています[4]。

## 木野 泰之

はじめまして一年理科一類の木野泰之です。

パソコンは今まであまり下ことがないのでこれから学んでいこうと思います。パソコンは購入予定があるのでこれからがたのしみです。早く一人前にパソコンを使いゲームプログラミングのような高度なことができるようになりたいです。

よろしくおねがいします。

[4]編注:うぅっ。よい時代になったものよのぉ(苦笑)

## 福林 一平

**ハンドル名** 文殊壱平 (Mont-Jeux Ippei)
**偽名** (申請中)
**仮の名** 福林一平 (FUKUBAYASHI Ippei)
**仮名** あいうえおアイウエオ etc.
**科類・組** 理科１類 27 組 (フランス語クラス)
**趣味** 知りたいですか？ ほんとに？
**持病** 自覚なし
**特記事項 1** 男色は好きではないかかわらず、マウスのボールをよく掃除する。
**特記事項 2** 福岡生まれのせいか、無言留守電が大嫌いである。
**特記事項 3** 特になし
**E-mail** ippei@t3.rim.or.jp,g741151@komaba.ecc.u-tokyo.ac.jp

## 本多 祐樹

パソコン歴が一週間の本多です。自分は初心者なのに、まわりの友達にスゴイ奴がいるのでいろいろへんなことを覚えています。

今、気になっていることは ”自炊” です。誰かおいしくて安い料理を知っていたらおしえてください。

よろしく！

## 高野 聡子

はじめまして！ 高野聡子 (たかのさとこ) と申します。

コンピューターは、大学に入って初めて使うので今はまだ、メールの出しかたと、読み方しかわかりませんが、早くうまくなりたいので、宜しくお願いします。

好きなことは、マンガをよむこと、音楽をきくこと、おかしをつくること、そして食べること、ボーーーッとすること、おしゃべりすること、ファミコンすることなどです。ファミコンは、落ちものや RPG が好きですが、はっきりいって、ヘタクソです。忘れていましたが、理 II の 11 組、学生証 NO. は、750317C です。

それでは、さようなら。

## 森田 雅樹

自己紹介です。。。。

---

もりたくんはげーむがすき。げーむつくりたい。ぐらふぃっかーしぼうなのでよろしく。
おんがくもやりたいけど、MIDIがたかいからとうぶんさきです。
それよりさきにましんかわなきゃ。。。

---

えーと、森田です。「もりりん」と呼んでいただけばおっけーです。今はマシンがないです。ちなみに昔は TOWNS ユーザーやってました。

密かな野望として、「シナリオもデザインもグラフィックも音楽もプログラムも自分で担当してゲームを作る」というのがありますが、、、そんなことしたら多分死ぬのでやらないかもしれないです。

軟式野球のサークルと百人一首のサークルとかけもちしてるのでどの程度顔を出せるかわかりませんが、どうぞよろしく。

グラフィッカーと音楽の勉強は、ぢつはこれから始めます (^^;;

# 分科会活動報告

## DirectDrawでdemoを作る

1997年4月29日

わたる

## はじめに

MEGADEMO[1]などでよく使われているfire effectのサンプルを、DirectDrawで作ってみたので紹介します。やってることは単純ですので、DirectXプログラミングをこれから学ぼうという人は、ぜひこの解説を読んでみてください。

ソースコード全体と実行ファイルは私のホームページ[2]か、いぬxBBSのファイラー、駒場のTSG部室305などで手に入ります。

ではさっそく始めましょう。

## 必要なライブラリ

DirectDrawを使うには、ddraw.lib[3]dxguid.lib[4]winmm.lib[5]の3つのライブラリをリンクする必要があります。Makefileなりプロジェクトワークスペースなりを設定してください。

## 各種宣言

```
#include <windows.h>
#include <ddraw.h>
#include <stdlib.h>
```

[1]「メガデモってなに？」って人はMEGADEMO&MODs MiNiMiNi INFO page for Japanese（旧JAPAN HQ）のホームページを見てください。http://www.netlaputa.or.jp/ d5/megademo.html

[2]http://www.yk.rim.or.jp/~wataruk/

[3]ddraw.dllをロードするライブラリ。

[4]DirectX関連のGUIDが定義されているライブラリ。

[5]タイマーを使うのに必要なライブラリ。

```
//グローバル変数
HWND hWnd              = NULL;
HINSTANCE hInstance    = NULL;
BOOL bIsActiveApp      = FALSE; //アプリケーションがアクティブなとき TRUE
char *errmsg           = NULL; //エラーメッセージの保存用
char *lpszAppName      = "Fire Effect Demo ver.0.0.1";
char *lpszWindowTitle = "Fire Effect";
LPDIRECTDRAW        lpDD          = NULL;
LPDIRECTDRAWSURFACE lpFrontBuffer = NULL;
LPDIRECTDRAWSURFACE lpBackBuffer  = NULL;
LPDIRECTDRAWPALETTE lpPalette     = NULL;

//Fire Effect が内部で使用する変数
static LPDIRECTDRAWSURFACE lpSysMemSurface = NULL;
static unsigned char *buffer; //描画先メモリアドレス
static RECT rect; //BltFast 用

//プロトタイプ宣言
void init_app();
void clean_app();
BOOL main_loop();
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

DirectDraw を使うには windows.h と ddraw.h を include します。DirectX2 以前では、これらを include する前に`#define INIGUID` とする必要がありました。しかし DirectX3 以降では、代わりに dxguid.lib をリンクすればよいようになりました。

init_app() と clean_app() はその名の通り、このプログラムの初期化と終了処理を行う関数です。main_loop() は fire effect を実際に描く処理をします。

他のグローバル変数については使い方は後で説明します。

## まずは WinMain() から

```
int WINAPI WinMain(HANDLE hThisInstance, HANDLE hPrevInstance,
                   LPSTR lpszArgs, int nWinMode)
{
  MSG msg;
  hInstance = hThisInstance;

  //初期化ルーチンを呼び出す
  init_app();

  while(TRUE){
    //メッセージループ
    while(PeekMessage(&msg, NULL, 0, 0, PM_REMOVE)){
      if(msg.message == WM_QUIT) goto FINISH;
```

```
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }

    //ESC が押されたら終了する
    if(GetAsyncKeyState(VK_ESCAPE) >> 1){
      PostQuitMessage(0);
    }

    //メインルーチンを実行
    if(bIsActiveApp)
      if(!main_loop()) goto FINISH;
  }

FINISH:
  //終了処理
  clean_app();
  return msg.wParam;
}
```

まず init_app() を呼び出しています。この関数は DirectDraw と fire effect の重要な初期化を行っており、次々節で解説します。

初期化が終わったらメッセージループが始まります。Windows アプリケーションは、たいていの場合メッセージの取得に GetMessage() を使いますが、ここでは PeekMessage() で行っています。GetMessage() はメッセージキューが空の場合、新しいメッセージが発行されるまで、処理が戻ってきません。ですから、ゲームやデモのようにリアルタイムな処理をしたいときは、メッセージが空でも処理が戻る PeekMessage() を使う必要があります。

メッセージをチェックした後に、ESC キーが押されていないか調べています。押されていればプログラムを終了させます。キーの状態を得るのに使っている GetAsyncKeyState() は、DirectX でゲームを作るときはよく使う API です。今回のプログラムでは、この API を使う必要はなかったのですが、例として紹介するためにわざと入れてみました。

Windows はアプリケーションに対し、キー入力もメッセージによって伝えてきます。しかしこの方式では、キー入力の検知と対応する処理の間に、時間差が生じてしまいます。一方 GetAsyncKeyState() は、この API が呼び出された瞬間のキーの状態を得るものです。これを使えば、キーが押された瞬間に、何か処理をする構造を作ることができます。

`GetAsyncKeyState(VK_ESCAPE) >> 1` という一見奇妙な処理は、戻り値の最下位ビットを消すためのものです。戻り値の最下位ビットは、指定されたキーが、最後に GetAsyncKeyState() が呼び出されてから一度でも押されていればオンになります。最下位ビットを無視するようにしないと、プログラムが起動直後に終了してしまう現象が、起きることがあります。

次に main_loop() を呼び出して、炎を描いています。アプリケーションがアクティブなときだけ main_loop() を実行するようになっています。フルスクリーンモード[6]のアプリケーションが非アクティブな状態で画面にアクセスすると、画面の解像度や色数が想定外のモードに変わっ

[6]画面全体を独占的に使えるモード。これを使うアプリケーションにウインドウはない。

ているので、一般保護違反を発生し強制終了させられてしまうのです。

このように、DirectDraw を使ったアプリケーションはアクティブ状態の監視が非常に重要です。次節でさらに詳しく解説します。

## メッセージプロシージャ WndProc()

```
LRESULT CALLBACK WndProc(HWND hThisWnd, UINT message,
                         WPARAM wParam, LPARAM lParam)
{
  switch(message){
  case WM_ACTIVATEAPP:
    bIsActiveApp = (BOOL)wParam;
    if(bIsActiveApp){
      //サーフェイスメモリの再確保
      if(lpFrontBuffer){
        if(lpFrontBuffer->IsLost())
          lpFrontBuffer->Restore();
        lpFrontBuffer->SetPalette(lpPalette);
      }
    }
    break;
  case WM_SETCURSOR:
    SetCursor(NULL);
    break;
  case WM_CLOSE:
    PostMessage(hWnd, WM_DESTROY, 0,0);
    break;
  case WM_DESTROY:
    PostQuitMessage(0);
      break;
  default:
    return DefWindowProc(hThisWnd, message, wParam, lParam);
  }
  return 0;
}
```

メッセージプロシージャではまず WM_ACTIVATEAPP をトラップし、アプリケーションのアクティブ・非アクティブ状態をグローバル変数に保存しています。その重要性は前節の最後で書きました。次に、アクティブ状態に切り替わったならば、サーフェイス[7]メモリの再確保を行っています。アプリケーションが非アクティブ状態になると、ビデオメモリ上のサーフェイスはメモリが解放されてしまうのです。これは他のアプリケーションのために解像度を戻したりするために、どうしても必要な措置です。従って非アクティブ状態からアクティブ状態に戻ったときは、ビデオメモリを再確保してやる必要があるのです。まず IDirectDrawSurface::IsLost[8]で

[7]画面用のメモリのこと。画面の全ドットの色情報が格納されている。
[8]DirectDrawSurface インターフェイスの IsLost() というメソッド、と言う意味の表記。

本当に解放されているか確認し、次に IDirectDrawSurface::Restore で元に戻します。Restore が実行されると、そのサーフェイスにアタッチされているサーフェイスも Restore されます。つまりフロントバッファ[9]が Restore されると自動的にバックバッファ[10]も Restore されるということです。

次に WM_SETCURSOR をハンドルして、マウスカーソルを消しています。フルスクリーンモードではマウスカーソルが正常に描画されないので、どうしてもこの処理が必要です。アプリケーションが Windows のマウスカーソルを使うことはできません。マウスカーソルは GDI によって描かれるものですが、GDI はフロントバッファとバックバッファのどちらか一方にしか描いてくれないのです。どちらのサーフェイスかは特定できません。画面のフリップ[11]によって、マウスカーソルが見えたり消えたりするようになってしまいます。一部のビデオカードでは、常に正常にマウスカーソルが見えますが、それはビデオチップがハードウェア的にマウスカーソルを表示する能力を持っているからです。もしフルスクリーンモード下でマウスを使いたい場合は、そのアプリケーションの責任でマウスカーソルを描かくようにしなければなりません。

## アプリケーションの初期化 init_app()

```
void init_app()
{
  srand((unsigned int)timeGetTime());
  rect.right  = 320;
  rect.left   = 0;
  rect.top    = 0;
  rect.bottom = 200;

  if(!create_black_window()) goto ERROR_EXIT;
  if(!create_ddraw())        goto ERROR_EXIT;
  if(!change_display())      goto ERROR_EXIT;
  if(!create_surfaces())     goto ERROR_EXIT;
  if(!create_palette())      goto ERROR_EXIT;
  if(!create_virtual_vram()) goto ERROR_EXIT;

  return;

ERROR_EXIT:
  clean_app();
  exit(1);
}
```

DirectDraw を使うためにはたくさんの準備が必要です。アプリケーションの初期化を行う

[9]ダブルバッファリング時の表画面。後述。
[10]ダブルバッファリング時の裏画面。後述。
[11]ダブルバッファリング時にフロントバッファとバックバッファを交換すること。

この関数 init_app() は、内部でさらにいくつもの初期化用関数を呼び出す構造になっています。

まず最初に乱数ジェネレータの初期化をしています。srand() は C 言語標準ライブラリの関数で、rand() が返す乱数のパターンを変えることができます。引数に指定している timeGetTime() は WIN32 API です。Windows が起動してから経過した時間をミリ秒単位で教えてくれます。この二つを組み合わせることにより、繰り返しアプリケーションが起動されても、異なる乱数を得られるようにできます。

次に RECT 構造体に画面の大きさを格納しています。画面全体を矩形領域転送するときに用います。

## 画面を黒で隠す

```
static BOOL create_black_window()
{
  WNDCLASS wcl;
  wcl.style         = 0;
  wcl.lpfnWndProc   = WndProc;
  wcl.hInstance     = hInstance;
  wcl.lpszClassName = lpszAppName;
  wcl.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
  wcl.hCursor       = NULL;
  wcl.hbrBackground = GetStockObject(BLACK_BRUSH);
  wcl.lpszMenuName  = NULL;
  wcl.cbClsExtra    = 0;
  wcl.cbWndExtra    = 0;
  if(!RegisterClass(&wcl)){
    errmsg = "ウインドウクラスの登録に失敗しました。";
    return FALSE;
  }

  hWnd = CreateWindow(
        lpszAppName, lpszWindowTitle,
        WS_POPUP | WS_VISIBLE | WS_SYSMENU,
        0,0,
        GetSystemMetrics(SM_CXSCREEN),
GetSystemMetrics(SM_CYSCREEN),
        HWND_DESKTOP, NULL, hInstance, NULL );
  if(!hWnd){
    errmsg = "ウィンドウを表示できません。";
    return FALSE;
  }

  return TRUE;
}
```

画面を初期化するのにあたって一番初めにすべきことは、画面全体と同じ大きさの黒いウインドウを作り、画面を隠してしまうことです。DirectDraw の初期化を始めると、解像度やパレットが変わるので、画面が一時的に見苦しくなってしまいます。しかし黒いウインドウを表

示しておけば、ユーザーはそれに気づきません。アプリケーションの終了時にも同様の効果が得られます。

## DirectDraw オブジェクトの生成

```
static BOOL create_ddraw()
{
  HRESULT hr;
  hr = DirectDrawCreate(NULL, &lpDD, NULL);
  if(hr){
    errmsg = "DirectDrawCreate() に失敗。";
    return FALSE;
  }
  return TRUE;
}
```

DirectDraw オブジェクトへのポインタは DirectDrawCreate によって得ます。第一引数には生成したい DirectDraw の GUID[12]を与えるのですが、NULL を指定しておけば自動的に使用中のディスプレイドライバが用いられます。

## 画面モードの変更

```
static BOOL change_display()
{
  HRESULT hr;
  hr = lpDD->SetCooperativeLevel(hWnd,
               DDSCL_EXCLUSIVE  | DDSCL_FULLSCREEN |
               DDSCL_ALLOWMODEX | DDSCL_ALLOWREBOOT    );
  if(hr){
    errmsg = "フルスクリーンモードにできません。";
    return FALSE;
  }
  hr = lpDD->SetDisplayMode(320, 200, 8);
  if(hr){
    errmsg = "320x200 8bpp モードを使用できません。";
    return FALSE;
  }
  return TRUE;
}
```

横方向解像度が 320 ドット以下のモードを使うには、協調レベル[13]に MODE X[14]の使用許可を指定する必要があります。一部のビデオカードでは指定しなくても大丈夫ですが、一般的ではありません。また、320x200 の解像度をサポートしていないビデオカードも存在していますので、モード変更に成功したかどうか必ずチェックします。

[12] Global Unique IDentifier 128 ビット整数で、Windows の各種オブジェクトの識別に用いられる。
[13] 他のアプリケーションに対してどれくらい排他的に DirectDraw を動作させるか調節するもの。
[14] 本来は VGA の用語だが、DirectDraw は 320 ドットのモードを一律にこう呼ぶ。

## フロントバッファとバックバッファの生成

```
static BOOL create_surfaces()
{
  HRESULT hr;
  DDSURFACEDESC ddsd;
  DDSCAPS ddscaps;

  //サーフェイスの作成
  ZeroMemory(&ddsd, sizeof(DDSURFACEDESC));
  ddsd.dwSize            = sizeof(DDSURFACEDESC);
  ddsd.dwFlags           = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
  ddsd.ddsCaps.dwCaps    = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |
                           DDSCAPS_COMPLEX |  DDSCAPS_VIDEOMEMORY;
  ddsd.dwBackBufferCount = 1;
  hr = lpDD->CreateSurface(&ddsd, &lpFrontBuffer, NULL);
  if(hr){
    errmsg = "サーフェイスの作成に失敗しました。";
    return FALSE;
  }

  //BackBuffer へのポインタを取得
  ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
  hr = lpFrontBuffer->GetAttachedSurface(&ddscaps, &lpBackBuffer);
  if(hr){
    errmsg = "BackBuffer の取得に失敗しました。";
    return FALSE;
  }

  return TRUE;
}
```

DirectDraw はダブルバッファリング機能をサポートしています[15]。ダブルバッファリングとは、表示用と描画用の二つの画面を用意して、一方を表示している間に他方に新しい画像を作成し、描画を完了したら画面を交換するというものです。描画過程が表示されないので、スクロールなどが滑らかに見えるようになる効果があります。DirectDraw では、表示中の画面をフロントバッファ、描画用の画面をバックバッファと言います。この二つは IDirectDraw::CreateSurface で同時に生成されます。得られるポインタはフロントバッファのものだけなので、IDirectDraw-Surface::GetAttacedSurface を使ってバックバッファへのポインタを別に得ます。

DirectX で API やインターフェイスに構造体をわたすときは、必ずゼロクリアして、dwSize というメンバに構造体のサイズを入れておかなけらばなりません。DirectX は 97 年 4 月現在ですでに version 5 beta1 までリリースされており、下位互換を保つための判定が構造体の大きさによって行われているのです。

[15] トリプルバッファリング以上も可能。

## パレットの生成

```
static BOOL create_palette()
{
  HRESULT hr;
  DDPIXELFORMAT ddpf;

  //パレットの作成。
  ZeroMemory(&ddpf, sizeof(ddpf));
  ddpf.dwSize = sizeof(ddpf);
  hr = lpBackBuffer->GetPixelFormat(&ddpf);
  BOOL use_palette = (ddpf.dwFlags & DDPF_PALETTEINDEXED8) ?
                     TRUE : FALSE;

  if(use_palette){
    PALETTEENTRY ppe[256];
    //パレットに色をセット
    set_fire_palette(ppe);

    //パレットの生成
    hr = lpDD->CreatePalette(DDPCAPS_8BIT | DDPCAPS_INITIALIZE,
                             ppe, &lpPalette, NULL);
    if(hr){
      errmsg = "DirectDrawPalette を作れません。";
      return FALSE;
    }
    //パレットの登録
    hr = lpBackBuffer->SetPalette(lpPalette);
    if(hr){
      errmsg = "SetPalette に失敗。";
      return FALSE;
    }
    hr = lpFrontBuffer->SetPalette(lpPalette);
    if(hr){
      errmsg = "SetPalette に失敗。";
      return FALSE;
    }
  }
  else{
    errmsg = "8bit パレットをサポートしていません。";
    return FALSE;
  }

  return TRUE;
}
```

まず、バックバッファが 8bit パレットをサポートしているが調べています。次に、set_fire_palette() を呼び出して PALETTEENTRY に炎らしい色をセットし、フロントバッファとバックバッファに同じパレットを登録しています。DirectDraw ではサーフェイスごとにパレットをかえたり共有したりできます。

set_fire_palette() では以下のように色を設定しています。

```
static void set_fire_palette(LPPALETTEENTRY pal)
{
  int i;
  for(i = 0; i < 128; i++){
    pal[i+128].peRed   = 255;
    pal[i+128].peGreen = 255;
    pal[i+128].peBlue  = 255;
    pal[i+128].peFlags = PC_RESERVED;
  }
  for(i = 0; i < 32; i++){
    pal[i+96].peRed    = 255;
    pal[i+96].peGreen  = 224 + i;
    pal[i+96].peBlue   = 8 * i;
    pal[i+96].peFlags  = PC_RESERVED;
  }
  for(i = 0; i < 64; i++){
    pal[i+32].peRed    = 128 + 2 * i;
    pal[i+32].peGreen  = 32 + 3 * i;
    pal[i+32].peBlue   = 0;
    pal[i+32].peFlags  = PC_RESERVED;
  }
  for(i = 0; i < 32; i++){
    pal[i   ].peRed    = 4 * i;
    pal[i   ].peGreen  = i;
    pal[i   ].peBlue   = 0;
    pal[i   ].peFlags  = PC_RESERVED;
  }
  return;
}
```

## 仮想 VRAM の作成

```
static BOOL create_virtual_vram()
{
  HRESULT hr;
  DDSURFACEDESC ddsd;
  ZeroMemory(&ddsd, sizeof(DDSURFACEDESC));
  ddsd.dwSize         = sizeof(ddsd);
  ddsd.dwFlags        = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
  ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN |
                        DDSCAPS_SYSTEMMEMORY;
  ddsd.dwWidth        = 320;
  ddsd.dwHeight       = 200;
  hr = lpDD->CreateSurface(&ddsd, &lpSysMemSurface, NULL);
  if(hr){
    errmsg = "CreateSurface に失敗。";
    return FALSE;
  }
  return TRUE;
```

```
}
```

仮想 VRAM とは、システムメモリ上に作成されたビデオメモリのコピーのことです。

本来の使い方は、ダブルバッファリングがサポートされていないビデオカードでも、仮想 VRAM の内容をビデオメモリに転送することにより、擬似的にダブルバッファリングを実現することです。使用中のビデオカードがダブルバッファリング可能かどうかは、DirectDraw が自動的に判断するので、プログラマはそのようなことを考慮する必要はなくなりました[16]。

しかし DirectDraw の便利な機能に頼りきるのはまずいのです。ビデオカードがハードウェア的にダブルバッファリングに対応している場合、バックバッファはビデオメモリ上に作成されます。ところが、CPU がビデオメモリにアクセスするには、PCI バスを経由しなければならないので時間がかかります。結果として、画面に読み書きを激しく行うプログラムは、システムメモリ上に画面を作成して、最後にそれをビデオメモリ上のバックバッファへ転送する方が速くなるのです。

fire effect は画面の読み書きが非常に多いので、仮想 VRAM 方式を採用することにしました。しかしデモやゲームを作る場合でも、いつもこの方式が速いとは限りません。スプライトや多重スクロールのように、画面の一部分を重ね合わせる処理はビデオメモリ上で行った方が速いからです。重ね合わせの処理は 64 ビットのビデオチップが行い、CPU には負荷がありません。描画用サーフェイスをどのメモリに作るかは、プログラムの内容により判断する必要があります。

## アプリケーションの終了処理 clean_app()

```
#define RELEASE(x) if(x){x->Release(); x=NULL;}
void clean_app()
{
  //画面モードの復帰
  if(lpDD){
    lpDD->FlipToGDISurface();
    lpDD->RestoreDisplayMode();
    if(hWnd)
      lpDD->SetCooperativeLevel(hWnd,DDSCL_NORMAL);
  }
  RELEASE(lpSysMemSurface);
  RELEASE(lpBackBuffer);
  RELEASE(lpFrontBuffer);
  RELEASE(lpPalette);
  RELEASE(lpDD);
  if(hWnd){
    DestroyWindow(hWnd);
```

[16]今日ほとんどのビデオカードはダブルバッファリングをサポートしていますが、なんと Millennium は対応していません。

```
    hWnd = 0;
  }
  if(errmsg)
    MessageBox(0, errmsg, "Fire Effect", MB_ICONERROR | MB_OK);
  return;
}
```

clean_app() はプログラムを終了させるために、いろいろな後始末を行っている関数です。まず DirectDraw を使って画面モードを元に戻してから、各種 DirectDraw オブジェクトを解放しています。次に、初期化や終了処理中の画面を見られないようにするために作った、真っ黒のウインドウを破壊しています。最後に、もしエラーメッセージがあればダイアログボックスを表示させています。

## 描画処理 main_loop()

### 炎を描くためにサーフェイスをロックする

```
BOOL main_loop()
{
  HRESULT hr;
  DDSURFACEDESC ddsd;
  ZeroMemory(&ddsd, sizeof(DDSURFACEDESC));
  ddsd.dwSize      = sizeof(ddsd);
  hr = lpSysMemSurface->Lock(NULL, &ddsd,
                          DDLOCK_SURFACEMEMORYPTR | DDLOCK_WAIT, 0);
  if(hr){
    errmsg = "Lock に失敗。";
    return FALSE;
  }
  buffer = (unsigned char *)ddsd.lpSurface;

  make_fire_source();
  draw_fire();

  lpSysMemSurface->Unlock(buffer);
  if(hr){
    errmsg = "Unlock に失敗。";
    return FALSE;
  }

  hr = lpBackBuffer->BltFast(0, 0, lpSysMemSurface, &rect,
                             DDBLTFAST_NOCOLORKEY | DDBLTFAST_WAIT );
  if(hr){
    errmsg = "BltFast に失敗。";
    return FALSE;
  }
```

```
  hr = lpFrontBuffer->Flip(NULL, DDFLIP_WAIT);
  if(hr){
    errmsg = "Flip に失敗。";
    return FALSE;
  }

  return TRUE;
}
```

サーフェイスメモリは一種のグローバルメモリです。複数のプログラムで共有することがあり得ます。従ってメモリを読み書きする前と後に排他制御を行う必要があります。それがIDirectDrawSurface::Lock と IDirectDrawSurface::Unock です。ロックしている間、アプリケーションの処理はクリティカルセクション扱いとなります。Windows 95 の場合は、WIN16 レベルでのロックがかかりますので、その間はマウスの動きがきごちなくなったり、デバッガのブレークポイントが無効になったりと、かなり強力な副作用があります。従ってサーフェイスをロックしている時間は、最小限に留めるようにしなければなりません。

main_loop() では、まず仮想 VRAM のサーフェイスをロックし、炎を描くために make_fire_source() と draw_fire() のふたつの関数を実行してから、ロックを解除しています。次に仮想 VRAM をバックバッファに転送して、画面をフリップさせています。

## 火種をまく make_fire_source()

```
static void make_fire_source()
{
  unsigned char *target = buffer + 320 * 199;
  for(int x = 0; x < 320; x++){
    int r = rand() & 127;
    if(r) *target = (unsigned char)(r + 128);
    else  *target = 0;
    target++;
  }
  return;
}
```

画面の一番下のラインにランダムな明るさの点を描いています。これらの点が、明るさを変えながら画面を上っていくというのが、fire effect の基本です。

描画先のアドレスを計算するときに、画面の幅を 320 バイトに決めうちしていますが、実はこれは DirectDraw のルールを破っています。画面の幅とサーフェイスメモリの幅が、必ずしも一致するとは限らないからです。幅のバイト数はビデオチップ側のアライメントの制約を受けます。4 バイト境界、8 バイト境界、16 バイト境界など、実際にどんな制限が課せられるかは、ビデオチップの種類によるので、実行時にしかわかりません。

今回のプログラムで私が 320 ドットに決めうちすることにしたのは、単なる手抜きです。システムメモリ上にあるので 4 バイト境界に従っていれば大丈夫だろうという、勝手な読みもあ

ります。

## 炎を舞い上げる draw_fire()

```
static void draw_fire()
{
  unsigned char *target_line = buffer + 320 * 60;
  for(int y = 0; y < 140; y++){
    unsigned char *target = target_line;
    for(int x = 0; x < 320; x++){
      int v = *target;
      if(y < 139) v += *(target + 319) + *(target + 321);
      else        v += 256;
      if(y < 137) v += *(target + 320 * 3);
      else        v += 128;
      v = (v - 20) / 4;
      if(v < 1)   v = 0;
      if(v > 255) v = 255;
      *target = (unsigned char)v;
      target++;
    }
    target_line += 320;
  }

  int count = (140*320 - 3*320) / 4;
  int src   = (int)buffer + 60*320 + 3*320;
  int dest  = (int)buffer + 60*320;
__asm{
  cld
  mov  ecx,  count
  mov  edi,  dest
  mov  esi,  src
  rep  movsd
  }

  return;
}
```

fire effect では画面を更新するとき、各点の明るさを、その点と、それより下にある 3 つの点の平均値に書き換えているのです。3 つの点の選び方により、炎の感じはだいぶ変わります。また、4 つの平均を取るところがポイントで、これにより割り算をビットシフトだけで済ませられるのです。

各点の更新を終えた後、インラインアセンブラで画面全体を 3 ドット下にスクロールさせています。この処理が今回のデモのこつで、炎のメラメラ感を出すのに貢献しています。

## 終わりに

実行画面を部報に載せられればよかったのですが、キャプチャーがパレットの取得に失敗し、あきらめざるを得ませんでした。そもそも fire effect の美しさは静止画ではわからないので、Windows を持っていない人もぜひ部室の 305 で見てみてください。

デモを Windows で動かすなんて、つい最近までは冗談にもならないくらい難しいことでした。しかし、DirectDraw の登場でその状況も変わりました。実際に、DOS でなく Windows で動くメガデモが現れ始めています。DirectSound を使った MOD プレイヤが公開されましたし、世界の#coders も徐々にですがプラットフォームを Windows に移行しそうです。

このプログラムの作成にかかった時間はわずか半日です。本当に簡単な作業でした。Windows プログラムのくせに、実行ファイルの大きさが 19 キロバイトしかないことからも、それがわかるでしょう。DirectX をこれから勉強したいという方は、ぜひこのソースファイルを解析してみてください。極力きれいに書きましたし、大きさも 9 キロバイトしかありません。何かわからないことがあれば、私の主催する DirectX 分科会で質問してみてください。

## 参考文献


- hacker'z alliance プログラミングとテクノに関するページ
  http://home.interlink.or.jp/~deimo/
- megademo mod otaku – Explain megademo effect
  http://www.urban.or.jp/home/ikeuchi/effect.htm
- #coders 256 byte fire compo
  http://www.ar.com.au/~gaffer/compo/compo.html

# 計算物理事始め

うえ

## 理論科学分科会について

先にメールでも告知しましたが、もう一度理論科学分科会の概要について説明しましょう。

- 理論科学分科会は基本的に週一回のペースで行う。
- 内容は、まず 1 週目にテキストを輪読（担当者を決め、その人が説明し、他の人はそれにつっこみを入れる）する。その内容について 2 週目までに各自プログラムを組み、2 週目に持ってきて発表してもらう。このサイクルを繰り返す。
- 使用するテキストは「偏微分方程式の差分解法」である。
- プログラミングの講習などは行わないため、ある程度のプログラミングの知識は前提とする。

とりあえず、連載[1]第一回はいろいろ基礎的な事に絞って書いていきます。

## 近似法あれこれ

### オイラー法

#### 概要

計算機で微分方程式を解く際には当然近似が必要となりますが、その際、もっとも簡単かつ分かりやすいのがここで示すオイラー法です。ただし、簡単なだけに解の精度はまるっきりなく、あっという間に誤差が蓄積してしまいます。始めて数値計算をする人の練習用と言えるでしょう。

微分方程式

$$\frac{dx(t)}{dt} = f(x,t)$$

について考えてみます。この形を見ればすぐにだれでも思いつくでしょうが、この式を数値的に解くいちばん簡単な方法は、

[1]続くのか？

$$\Delta x = f(x,t)\Delta t$$

として、無限小の時間を有限の（しかし、十分に小さい）時間で置き換えてしまい、

$$x(t+\Delta t) = x(t) + f(t)\Delta t$$

としてしまう事でしょう。この方法をずっと繰り返せば任意の時刻における $f(x,t)$ の値が定まります。この方法をオイラー法と呼んでいるわけです。

この方法は $x(t)$ の将来をテイラー展開の一次の項までで予測してしまおう、という手法をとっているため、当然解の精度はひどいものになってしまいます。

## 実践

オイラー法を使って、ケプラー運動をシュミレートしてみましょう。太陽が動いては少し面倒になるので太陽の質量は十分重いため、太陽は原点に静止しているとします。惑星の従う方程式は、

$$\frac{d^2x}{dt^2} = -k\frac{x}{(\sqrt{x^2+y^2})^3}$$

$$\frac{d^2y}{dt^2} = -k\frac{y}{(\sqrt{x^2+y^2})^3}$$

となります。（ただし $k$ は正の定数）

このことを実際にプログラムしてみましょう。以下に一例をあげます。

```
// ケプラー運動

#include <iostream.h>
#include <math.h>

#define K 1.0
#define dt 0.01

int main()
{
 double x,y, vx,vy, ax,ay, R;

 cout << " input x,y " << endl;
 cin >> x >> y;
 cout << "input vx,vy" << endl;
 cin >> vx >> vy;

 while(1)
 {
   R = sqrt(x*x+y*y);
```

```
  ax = -k*x/(R*R*R);
  ay = -k*y/(R*R*R);
  vx = vx + ax*dt;
  vy = vy + ay*dt;
   x =  x + vx*dt;
   y =  y + vy*dt;

  cout << x << y << endl;
 }
 return 1;
}
```

プログラムはこんなに簡単なんです。

グラフィック関係の命令は各自で補完しましょう。

ケプラー運動は頑張れば解析的に解けますし、それほど面白くもありませんが、多体問題に拡張してみたら面白いのではないのでしょうか。

## ルンゲクッタ法

### 改良オイラー法

しかし、オイラー法は精度が悪すぎます。もっといい近似法を使いましょう。

オイラー法はテイラー展開の 1 次までで打ち切っています。ここで、もっと高次の項まで一致させる方法を考えます。

解析すべき方程式は

$$\frac{dx(t)}{dt} = f(x,t)$$

でした。

テイラー展開

$$x(t+h) = x(t) + hf(x,t) + \frac{h^2}{2!}f'(x,t) + \cdots + \frac{h^n}{n!}f^{(n-1)}(x,t) + \frac{h^{n+1}}{(n+1)!}f^n(x,\xi)$$

を 2 次で打ち切ると

$$x(t+h) = x(t) + hf(x,t) + \frac{h^2}{2!}(f_t + f_x f)$$

となります。ここで、$f_t$、$f_x$ などは計算せず、$(x,t)$ と $(x(t+h),t+h)$ の間の $g$ の値によってこれらの代用をするのがルンゲークッタ法の考え方です。つまり、適当な定数 $c_0, c_1, p, \alpha$ を選んで

$$k_0 = hf(x,t), k_1 = hf(x+\alpha k_0, t+ph), x(t+h) = x(t) + c_0k_0 + c_1k_1$$

この $x(t+h)$ が２次の項まで一致するように選べばいいわけです。

$$k_1 = hf(x,t) + h^2(\alpha f_x(x,t)f + pf_t(x,t)) + O(t^3)$$

なので

$$x(t+h) = x(t) + (c_0+c_1)hf(x,t) + c_1h^2(\alpha f_x(x,t) + pf_t(x,t)) + O(h^3)$$

従って定数の満たすべき条件は

$$c_0 + c_1 = 1, c_1p = \frac{1}{2}, c_1\alpha = \frac{1}{2}$$

です。（この係数の選び方にはもちろん自由度があります）

$$p = \alpha = \frac{1}{2}, C_0 = 0, C_1 = 1$$

としたものを「改良オイラー法」といいます。
まとめると

$$k_0 = hf(x,t), k_1 = hf(x + \frac{k_0}{2}, t + \frac{h}{2})$$
$$x(t+h) = x(t) + k_1$$

です。

### ４次のルンゲ・クッタ法

同じような方法でテイラー展開の４次の項まで一致する公式を作ると

$$k_0 = hf(x,t), k_1 = hf(x + \frac{k_0}{2}, t + \frac{h}{2}$$
$$k_2 = hf(x + \frac{k_1}{2}, t + \frac{h}{2}), k_3 = hf(x + k_2, t + h)$$
$$x(t+h) = x(t) + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3)$$

となります。
普通は、4 次まで一致するように作ったものをルンゲクッタ法と呼んでいます。

## シンプレティック数値積分法

以上で述べた方法は、いずれもテイラー展開を途中で打ち切り、現在の値から未来を予測しようとしたものでした。これではどうやっても誤差が集積してしまうのは明らかでしょう。

ところが、「エネルギーが保存される」近似法というものがあるのです。それを「シンプレティック数値積分法」といいます。しかし、これに関してはかなり複雑な事を述べなくてはなりません[2]。従ってここでは省略しますが、知りたいという人は TSG のホームページを辿って NAO さんのホームページへと行って下さい。そこに詳しい資料があるはずです。

## 偏微分方程式の差分解法

独立変数が 1 つだった常微分方程式に対して、偏微分方程式には独立変数が 2 つ以上あります。従ってこれまで述べてきた方法をふまえ、新たな方法を確立せねばなりません。解き方としては、

- 差分法
- 有限要素法
- スペクトル法

等がありますが、この分科会では最も親しみやすいであろう差分法を扱います。

差分法の基本的な考えは、名前のとうり座標による偏微分を差分で置き換えてしまおうと言う事です。詳しい事は分科会でやりますので、興味のある人はぜひ参加して下さい。

ただ、差分法は理解しやすいですが、解の精度がそれほどよくはないので、非線形項を含む方程式を解析する際には、発散したりする事もままあります。これに満足できない人は、自分で他の方法も探求してみましょう。

[2]解析力学の知識も必要

# 一般記事

## 3x5 と いぬ x

おざわ x

## アンブレラ@神泉について

おざわ x%有限会社アンブレラだいひょーとりまりやくしゃちょー

おととしまでは現役 TSG 人だったおざわ x です。きょねんは大学の研究生とゆーわけで、現役と OB の境目にそれとなくふらふらしていたよーな気がします。このあやふやさがよくもわるくも TSG の特徴ですね。

ぼくの同期とその前後 1 年くらいの TSG メンバーがけっこー社員なアンブレラという会社をきょねんつくりました。今は任天堂 64 とゆー家庭用ゲーム機向けのゲームをつくっています。そのほかいろいろあやしいこともしていなくもないというような。まあ、つくったばっかの会社なのでいろいろ方向性を模索しながらやってます:>。

現在事務所が神泉駅からすぐのところにあって、駒場東大からかなり近いです。学館 305 からだと歩いてあたりまえの距離。

事務所内に学生と共生するためのスペースをつくっている最中で、ここは 3x5 とよんでいます。3x5 にはコンピューターが数台並べて、305 よりはプログラミングしやすい環境になる (はず:>) です。

プログラミングの勉強をしよーってときには利用していいよ:)。そのかわり誕生日をおしえてね (謎。

でわ神泉 (か吉祥寺か渋谷か仙川か下北沢か…) でおあいしましょー。
…でしょしょ？

## いぬ xBBS について

おざわ x%いぬ xBBS しすおぺ

いぬ。BBS という草の根 BBS が昔あって、TSG の活動の一部はこの BBS 上でおこなわれ

ていたような感じでした。ったく☆って友人が長らく管理していたのですが、残念なことに彼は就職して遠くにとばされてしました。

さてどーするかねーということになって、ぼくがひきとることになりました。むかしぼくは「おざわ。」というハンドルだったのをいまは「おざわ x」にしているので、いぬ。は暫定的にいぬ x という名称に変えて運用のテスト中です。いつになったら正規なのかとか、正式名称はいつきまるのかとか、そういう細かいことは気にせずに、TSG の連絡事項なんかにこの BBS を利用してやってくださいませ。

ところで、いぬ x 自身には明確な目的なんかがあるわけではなくて、そのときそのときにアクティブな人がやりたい活動に利用してほしいなーというのが管理者の願いです。

```
いぬ xBBS 03-5457-3405 1200/2400/9600/14400/28800(V.34/V.FC)bps 24hours
```

…でしょしょ？

## なぜなに 3x5

すーゆー

## 3x5

今回 おざわ x さんに説明頂いた 3x5 ですが、新しいマシンを購入するなど現在環境を整備中です。夏頃までには分科会等で使える環境にしたいですね。305 の環境に不満の方は期待していてください[1]。

ただし、この部屋に関しては部室のように出入りが自由というわけにはいきません。アンブレラ事務所内にあるので、セキュリティーは万全に近いですが、アンブレラの方々に迷惑のかからないようううまくやっていく必要があります。現在、具体的には次の章のような利用規約を作ってあるので、特に 2 年の人は覚えておいてください。

また、最大同時利用人数の制限をどうするかという問題が残っていますが、ま、それは会議を開くなりしましょう。

[1] 305 にもマシン環境の変更が予定されています。おそらく MMXPentium200MHz ぐらいのマシンが 305 に設置されるでしょう

# 3x5 利用規約

アンブレラ事務所内スペース利用規約
==========

この規約は該当スペースを利用する全ての TSGer に適用される。

■ 対外的規定

○連絡
・事前に電話にて要連絡
・電話にて連絡する人物はアンブレラに面識のある人物であること

○現地
・アンブレラに迷惑をかける行為 (騒ぐ等) は厳禁
・該当スペース内に置かれているものでもアンブレラの機材は使用禁止 (ゲーム機など)

■ アンブレラ委員

○条件
・役員交代後の１年生 or ２年生 or 駒場に在学している人 であることが必要条件
・明確な区別のために登録が必要
・登録の更新は委員が会議にて決定する

○義務
・月１回ペースで定期的に会議を行う
・アンブレラの方々との窓口となる用に努めること

○役割
・利用の連絡を行うことが出来る
・その他 該当スペースを利用するに必要な事項の決定権を持つと同時にこの利用規約を改変する権利を持つ

■ TSGer 利用方法

○事前連絡
・アンブレラ委員は自身で連絡を入れることが出来る
・非アンブレラ委員はアンブレラ委員に連絡を入れてもらった上で必ず同委員と同伴のこと
・３年以上でアンブレラと面識がある人に限り、直接連絡を入れることができ、アンブレラ委員への通告は不要である

○責任者
・連絡を入れた人が責任者となる
・責任者は入室／退室ノートに記されている項目を記録すること
・責任者は責任者たりうる (事前連絡を入れる権限を持つ) 人以外を残して退室することはできない

○現地での規則
・アンブレラの機材 (ゲーム機やＴＶ等) を使用することは厳禁
・ゲーム機持ち込み禁止
・アンブレラに迷惑をかける行為 (騒ぐ等) は厳禁
・宿泊禁止

・ごみ箱は設置していないのでゴミは基本的に持ち帰ること

○ TSGer の利用基本精神
・該当するスペースを「３×５」もしくはそれに類する表記をする
同時に「さんばつご」という読みを与える
・積極的に３Ｘ５を活用するように努力する
・分科会で使用することで活性化を図る

平成９年３月３０日 初版
平成９年４月１０日 改訂２版

==========

## アンブレラ委員

現在の [アンブレラ委員] は以下の通りです

| | | |
|---|---|---|
| 植原 洋介 | うえはら ようすけ | うえ |
| 黒川 秀樹 | くろかわ ひでき | |
| 坂本 崇裕 | さかもと たかひろ | すーゆー |
| 菅原 豊 | すがわら ゆたか | |
| 丹下 吉雄 | たんげ よしお | |
| 花岡 昇平 | はなおか しょうへい | ヴぇあ |
| 坂東 大五郎 | ばんどう だいごろう | ばんだい |

わかんないことが有ったら、上記の人達に聞いてね☆

**編集後記**

やべー。もう AM6:30 じゃないか。うーん、編集する時間が毎回遅くなってきてしまっている。これはいかんなぁ(笑)。

**Beep! 業務連絡**

情報教育棟活用分科会は、

毎週木曜 6 限 (6:00 〜 7:30) @ 学生会館 305

に事実上決定です。6:10 になってもすーゆー君が現れなかったら休講ですので、よろしく。うぉー、今日やる内容まだ決めてないよう (T_T) さっそく休講か ?!

**Beep! Beep! 業務連絡 II**

TEX 分科会は人数が少ない&担当者が多忙なため、特に要望がない限り情報教育棟活用分科会に吸収される可能性があります。

ただし、希望があれば情報教育棟だけでなく 3x5 を利用した演習をやることも考えられます。

3x5 に行ってみたいみたいという人は、DirectX 分科会か C++ 分科会か TeX 分科会に入るかもしくはアンブレラ委員 (事実上 2 年生全員) に声かけてみるなりしてください。

**理論科学グループ　部報 207 号**

1997 年 5 月 10 日　発行

発行者　植原 洋介

編集者　坂本 崇裕

発行所　理論科学グループ

〒 153　東京都目黒区駒場 3–8–1

東京大学教養学部内学生会館 305

Telephone: 03–5454–4343

**理論科学グループ部報 第 207 号**

**— 新入生歓迎号 —**

**1997 年 5 月 10 日**

*THEORETICAL SCIENCE GROUP*