Theoretical Science Group

理論科学グループ

AT WORK

作業中

1997年3月8日すーゆー

部報 205 号
— 追い出しコンパ号 —

目　次

追い出しコンパ

追い出しコンパに寄せて

今年も追い出しの季節がやってきました。といっても 1 年生がこの場に立ち会うのは初めてのことではありますが (笑)。

現時点でコンパ委員が把握している今年の主賓の方々は 7 名です。かつての TSG (というか『あらかわ犬』かも...) の繁栄を築いてきた方々も含まれています。

TSG の場合、駒場から離れた段階で TSG としての大方の仕事からは逃れられるわけで、実感が湧かないこともあるかもしれません。しかし、今回のコンパで正式に追い出されることがその一助となれば幸いです。

皆様の今後の御活躍を期待しております

1996 年度追い出し役一同

中村 隆幸 さん

いぬ。BBS 閉局のお知らせ

ども。SysOp の、ったく☆、です。

3 年半の長きに渡ってご愛顧いただきました当いぬ。BBS は、このたび一身上の都合により、閉局させていただく運びとなりました。

会員のみなさまには (以下略)

具体的な閉局期日は、現在のところ未定ですが、大体 1997 年 3 月下旬を予定しております。詳しい日時が決定し次第、あらためてこの場で告知いたします。

閉局後の当 BBS の有形・無形の資産の取り扱いについて説明いたします。

まず、いぬ。BBS の運用資金として、みなさまから多額の寄付を頂けたことに感謝いたします。集まった資金は、月々の電気料金及び電話基本料金以外の経費である、回線設備や機材調達費に充当しました。現物援助を頂けたこともあって、結果的には寄付額と回線・機材調達費がほとんど等しくなりました。

BBS を運用する機材・回線設備等は、おざわ x (ozawa) 氏に譲渡することとし、機材を (有) アンブレラ（東京都渋谷区）事務所内に設置して、新たに BBS を運用していただく予定です。

新しい BBS の運用開始時期は未定ですが、1997 年 4 月上旬までには運用を開始していただけるのではないかと予想しています。その際、移転にともない電話番号が変更になります。新しい電話番号の告知は、一次的には italk (electra.is.s.u-tokyo.ac.jp:12345) 上で行うこととします。また、可能であれば、いぬ。BBS 上でも新 BBS の電話番号の告知を行います。

現在の会員データ、ファイルライブラリ等は全て消去して、まったく別の BBS として運用することになります。ただし Notes の書き込みについては、新しい BBS 上でログアーカイブの形で公開するかもしれません。詳しいことは未定です。

italk-server の運用に関するお知らせ

ども。italk-server 管理者の [ったく] です。

このたび無事大学院理学系研究科情報科学専攻を修了できそうな気配がただよっていますので、今後の italk-server の運用についてお知らせします。

現在 italk は、メインのサーバーを electra.is.s.u-tokyo.ac.jp:12345 にて、バックアップを ecc-lx00.hongo.ecc.u-tokyo.ac.jp:12345, mercury.st.rim.or.jp:12345 の二箇所で動作させています。また、ecc-as50.komaba.ecc.u-tokyo.ac.jp では、実質的なバックアップサーバーを、かも君が運用中です。

electra は益田研究室 (@is.s) のマシンですが、私のアカウントは修了後も残ることになるので、引き続き italk-server の運用を続ける予定です。ただし、今までと違い、障害発生時に十分なメンテナンスができなくなることが予想されます。

バックアップサーバーとして、mercury の italk-server を一応引き続き動作させる予定ではいます。しかし主なバックアップサーバーとして、ecc-lx00 またはその他適当なマシン上で、かも君に italk-server を運用してもらうことにします。

というのが半年程度の予定で、それ以降は、まあなるようになるでしょう (ぉ

そして…

サークルとは人の和であり、目的を持つ人のあつまりであるわけで、

そのなかで人のあつまる目的と手段とを、どちらも自分なりのやり方でまがりなりにも提供できて、それをひろく受け入れてもらえたというのは、とてもうれしいことであるなあとおもうわけなのです。

三瓶 均 さん

91 年 4 月 コンピュータ系のサークルに入ろうと思い、「槌音」を見て紹介文がまともだった TSG に入る。ここで TSG に入らなかったら人生は変わっていただろう。特に人知

研に入っていたら。(謎)

91 年度の前半 C 分科会で C を習う。それ以外の時間はただなんとなく部室にいたような気がする。先輩たちはなぜか「まぐろ」や「きす」など魚の話ばかりしていたので、さっぱり話についていけなかった。

91 年 11 月 駒祭向けに挟み将棋「挟むんです。」を制作。自宅のマシン PC-286V にはこの当時 HDD がついてなかったので、開発はフロッピー 2 枚の上の Quick-C で行われた。ちなみに、このマシンはまだ家にあるが、いまだに増設 RAM はついていない。残念ながら「挟むんです。」最終バージョンの行方は不明である。(おい

92 年度の前半 C 分科会を主催。主催者がバグ入りのサンプルを書き、なぜか周りにいる TEA 君やったく☆さんがそれを修正するという風変わりなものだった。それはそれとして、CYC にはリストの書き方くらいは教えたつもりである。

92 年 7〜8 月 コンパ委員として、伊豆・宇佐美での夏合宿を企画する。近年にない失敗だったと自負している。(爆) 生協のアンケートはがきに怒りをぶつけたところ、「温泉民宿　ふくだ」は翌年の生協のパンフレットから姿を消した。そういうわけで、苦情ははっきり言うのが次世代への義務と言えよう。

92 年 11 月 駒祭向けにコアウォーズのプログラムを制作。といっても骨格は TEA 君が作ったものであった。ルールを調べるのにえらく苦労した記憶がある。今ならサーチエンジン一発なんだろうなあ。

93 年 3 月 初めての自分専用マシンとして IBM-PC 互換機を買う。486DX (33MHz), 8MB RAM という構成は、当時としてはかなり強力なものだった。TeX をインストールして感動したり、Mandelbrot 集合を描かせて喜んだりしていた。

モデムを買ったのも確かこの頃で、以降パソコン通信への道が開けていく。Sportster は初期不良交換期限が過ぎてしばらくしてから壊れてくれたのだが。(泣

93 年 4 月 本郷に進学。教育用計算機センターの分散配置端末が S-3/80 だったので、いきなり UNIX の海に放り出される。日本語を使うには X を立ち上げるしかなかったが、このマシンパワーで X を使うのはかなりつらいものがあった。おまけに当時はネットワークも腐っていたので、凍ってしまってしかたなく電源ぶちとかは日常茶飯事であった。(おい

こんな苦しい状況でも Moria で遊んでいた記憶がある。サルだ。

93 年度の後半 根津研の会計であったのだが、運営がかなり行き詰まり、閉所やむなしということになる。結論として 94 年度で閉めよう、ということになったが、その後なんかわからないうちに存続することになった。しかし次期会計はどうなるんだ。＞今の根津研

93 年 12 月 唐突に MSX のゲームがやりたくなり、いぬ。BBS に「持ってたら貸して〜」という書き込みをする。この結果、305 で MSX ブームが起こったようだが結局私にソフトを貸してくれる人はいなかった。

94 年 3 月 春休みを利用してサルのように Nethack(version 3.1.3) をやった結果、Caveman で

クリアすることができた。この後の半年のあいだに、7 職業でクリアに成功する。

94 年 4 月 研究室に配属になる。PC-286L が余っていたので専用マシンとする。いやー 2M も RAM があれば V30 でもいけるねえ。(やや嘘

94 年 7 月 院試が近いので根津研の Final Fantasy 6 を大急ぎでクリア。(何やってんだ。)

95 年 2 月 卒論が煮詰まると fj.news.usage を読む。そんなすさんだ生活だった。

95 年 3 月 卒論も終わったので、CPU とマザーボードとビデオカードをグレードアップした。安かったので AMD DX4-100MHz にしたのだが、まさかこれを修論の作業マシンにするとは思わなかった。

96 年 3 月 学校に自分のマシンを持ち込み、UTnet に接続。専用線状態となる。このあと、italk の合間に修論研究をやる。修論のテーマは「ダイクストラの方法 (しかも未完成) を用いた修士号の奪取」である。(いちおう嘘

97 年 3 月 こうしてみると業績はなにもないですねー。しかしながら楽しめた 6 年間でした。まあ修了して就職するわけですが、今や部報も WWW 上で読めるし、italk はあるしで、TSG と遠く離れることはなさそうです。というわけで今後ともよろしくお付き合いください。

活動報告

スキー合宿の傾向と対策

愛シューター野村

参加した皆様：GANA、Nishi、NAO、げる、大岩、私（敬称略）

裏工作はしっかりと

スキー合宿と直接の関係はないんですが裏方さんは大変なのです。2 カ月位前から交渉を始めて何回も調整していきます。一番厄介なのが人数でほんとーに予想がつかなかったですね。今回は 15 人と見積もっていたんですが結局 6 人になってしまいました。MO の為に合宿をやめちゃいかんすよ、ぼそっ（笑）場所はもちろんレンタルやら保険、はては到着時の朝食まで決めます。パックツアーを使えばもっと楽に済むんでしょうが、やっぱ TSGer たる者カスタマイズにはこだわります。

教訓：根回しはきっちりやっときましょう

さあ行ってみよう

年が明けて 1 月 4 日、原宿で集合とあいなりました。旅行代理店に指定されてた場所は大量の人間がうごめいていて屋台まで出ている始末。原宿駅で 5 人揃ったのですが大岩氏だけここで合流しました。ちゃんと会えたのは神のお導きですね。

さて、とにかく手続きを済ませてバスを待つ、待つ、待つ、、、、、、こねーぞ、おい。年末年始は道路事情が悪いようでバスは 30 分以上遅れて到着しました。乗り込んで座ってしまえば後は寝てるだけで OK、着けば起こしてくれます。

教訓：集合場所はわかりやすい場所にしとこーね

着いてみれば雪国

おおっ、周り一面雪だらけ。関西人の私にはすんげーめずらしい光景です。しかし、しかーし、寒いぞこれは、いやマジで。旅館の地図を持つ手が震えてもう大変。それもそのはず着いたのは朝の 4 時、予定時刻より 2 時間も早いじゃねーか、どうなってんだよ？

「バスは夜の高速をひた走ったんじゃ、わしらそれはそれはおそろしゅうて、、、」［げる＆大岩氏談］

幸い旅館はバス停の近くにあり迷うことなく着けました。さすがにチェックインしようにもカウンターには人がいません、当然ですね。しかたなくロビーで時間を潰すことにしたんですが、寒いこと寒いこと。ストーブから離れた場所で寝ようとした NAO さんが諦めてました。で、レベル E やら富士見ファンタジア文庫やらが飛び交う中、無駄話をしていると宿の人がやってきて一言

「おんや、この時間玄関締めてるんじゃけどのぅ、いやぁあやうく凍死するところでしたなぁ」おいおい（笑）

なんでも旅館の若い衆がカラオケに行ってたおかげで開いていたそーです。

教訓：バスの運ちゃんには気をつけろ

☆うれしはずかし初体験☆

私にとって 10 数年ぶりのスキー、ほとんど初めてみたいなもんです。もうドッキドキ、うふ。ウエアと板を決めてリフト券を買い、いざゲレンデへ。私は白銀の貴公子、世界は私に向かって微笑みかける。るるる、、、進まねー、何だこの非人間的な道具はよー。リフトまでの坂がチョモランマの様にそびえ立って見えるぞ。なんとかリフトに乗り込んで中腹にたどり着いた瞬間、転ぶ、起き上がって転ぶ、滑り落ちて転ぶ。いやぁスキーって難しいスポーツだったんですね。華麗な GANA さん、豪快な Nishi さん、堅実な NAO さん、俊敏な大岩&げるコンビ、ターンすら出来ない私。

あまりの惨状に見かねた大岩＆げる氏が午前中ずっとコーチしてくれました。ちなみに他のメンバーは別れて別行動、すまんのー両氏、感謝してます。しかし、それでも、なおかつ、なぜかしらターンが出来ない。俺ってこんなに運動神経が悪かったのか（泣）。それでも何とか午後には滑れるようになりました。努力の成果って奴ですね。

経験則：なんて空は青いんだ

初夜のたしなみ

たしなもうにも皆疲れきってへとへと。部屋に転がり込むと同時に倒れ込んでしまいました。部屋自体は百人乗っても潰れないくらい広くって快適でしたが、夕飯を食べたら何もする気が起こらなかったですね。実際 NAO さんと私が布団を敷かなかったらそのまま寝てしまったでしょう。そういえば夕食の内容が他の客と違ってました。むー、旅行代金をケチるとこんな所にしわ寄せが来るのか。

教訓：うまいもの食いたきゃケチるな

2 日目のさわやかな朝です

と思ったらちょっと雪が降ってました。吹雪くというほどでもなく、スキーには差障りがなかったのは日頃の行ないの良さでしょう。気を良くした私達は 3 日目の朝も滑れるようにレンタル期間を延長しに行きました。これが悪夢の始まりになるとは誰が予想したでしょう。

そんなことはつゆ知らず、ずんずん山を登って行きます。

ずんずん、ずんずん、ずんずん

「ねっ、僕ここらへんでいいんだけど」（私）

「いや上の方がすいてて滑りやすいよ」（皆様）

ずんずん、ずんずん、ずんずん

「ふえー、高いっすよ」（私）

「だいじょーぶ、だいじょーぶ」（皆様）

ずんずん、ずんずん、ずんずん

「、、、、下界がみえねぇ」（私）

何とか滑っては止まり滑っては止まりしながら降りていきました。初日にコーチしてもらったおかげで転ぶ回数は減りましたが一度板が外れる程激しくこけて死にそうでした。親指つき指して痛かった、くすん。

しばらくすると初級と中級が別れてる所に着きました。

「僕、初級いきますから皆さんは中級いって下さい」（私）

「だいじょーぶ、やっぱ一回は難しい所を滑っておかないとね。うんうん、あれだけ滑べれりゃだいじょーぶだって」（皆様）

まぢで恐かったです。ちょっと斜面がきついように見えましたが、初級とあまり差はないようでした。が、実は斜面が凸凹で板が浮くんですよ。止まる瞬間に山に乗り上げてるとすぐにバランスを崩してこける or あらぬ方へ滑べっていく、ああぁ。この後人間不審に陥った私は一人初級を、NAO さんはスキー学校へ、他のメンバーは全コース制覇へとそれぞれの人生を歩

むことになります。あとで聞いた話によると上級コースは人知を越えた場所で、比較的人間よりだった Nishi さんには辛かったとのことです。私にはとても想像できません。

教訓：うかつに人を信じてはいけない

2 日目の夜はね

なんか 2 日目の夜は余力がありました。夕食も他の客より 1 品少ないようでしたが豪華でうまかったです。ボケボケの私は鍵を部屋の中に閉じ込めてしまって間抜けぶりを発揮してしまいした。鍵といえば金庫の鍵が抜けなくなるというアクシデントも発生。中に旅行代金を入れてたら帰ってこれなかったかも知れません。結局チェックアウトの時までに抜くことは出来ませんでした、はずかしいですねー。

くつろいでいると突然の電話がありバスの出発が 2 時間早まるとのこと。むー、せっかく延長したのに。それでも頑張って明日も滑ることになりました。野郎ばっかで華やかさには欠けましたが、やはり旅行は楽しいものです。みんなして夜遅くまで話し込んでいました。プログラムスタイルの話が出るあたりはさすがに TSGer。皆さん if の書き方はともかく、tab の数までこだわってプログラムを組んでたんですね。適当ーにコーディングしている私には目からウロコもんでした。

教訓：鍵の管理はしっかりやろーね

あっというまに最終日

さて朝飯もくったし滑べりにいこうかという矢先

「板が無い !! 」サザエさん的に表現するなら「んがんぐっ」ってな感じです。これには驚きました。誰かが間違えて持っていったのか、いや私達をはめようとする秘密結社の仕業か、もしかしたら怪事件に巻き込まれたのかも、明日の新聞一面は俺達が飾るのかあぁぁ、ぷちっ。怪しいのはレンタル屋のおやじだ、あいつ俺達を殺意のこもった眼差しで見つめていたぞ。まちがいない、あいつだ、あいつしかいない、殺られる前に殺ってしまぇ。俺達の未来を潰されてたまるか。

結局レンタル屋のおやじに聞いた所、間違って回収してしまったとのこと。借りたまま返さない客がいるのでこのように旅館に取りにいくのだそうです。無駄な時間を使ってしまったのでもう滑べる時間はありませんでした、もったいない。仕方なく清算をしようとすると、レンタル内容を書いた紙が無い。いや、私がなくしたんじゃないですよ。レンタル屋の担当者が間違って持って帰ったらしいんです。その上話を聞いてみると借りていた板は料金より安いモノ

らしいんです。くそっどーりで滑べりにくかったはずだ。怪しいのはレンタル屋のおやじだ、あいつ俺達を殺意のこもった眼差しで見つめていたぞ。まちがいない、あいつだ、あいつしかいない、殺られる前に殺ってしまえ。俺達の未来を潰されてたまるか。しつこい（笑）

当然代金を値切ろうと思ったら担当者がいないんでよく分からないとのたまう。担当者はひさびさの休日に浮き足だって帰っていったんだそうです（怒）結局残金は支払ってません、いいんじゃ!!

教訓：宮前スポーツには気をつけろ

あっというまに最終日 2

その様子を見ていた旅館の人々いわく

「だいたいフェニックス観光さんっていかんのよ。清算方法ややこしいしねぇ。21％もピンはねするんよ。私らの中では評判悪いやねぇ。」親父、おかみ、息子が声を揃えて言ってました。おいおい、そこまで聞いてないってば。

気をとり直してチェックアウト、バス乗り場に行きました。途中みやげもの屋に寄ってお買いもの、305 の人々はおこぼれにあずかれたかも知れませんね。集合場所に屋根がなかったので郵便局の軒先で待たせてもらいました。お客の人が中に入っていくたびに謝っていくので恐縮してしまいました。もちっとマトモな場所は無いんかい、まったく。そして来たバスはマイクロバス、ええっ、えあろすけるとんたいぷのりくらいにんぐしーとのバス（宣伝文句より）じゃなかったのか。さらに激しく回り道をして新宿に向かうらしい。もういいっす、煮るなり、焼くなり、勝手にして下さい、ぶちぶち。でも到着時刻だけはしっかり予定通りの 20:30 でした。

教訓：フェニックス観光にも気をつけろ

感想を一言

こんな感じでスキー合宿は終了しました。短い割にトラブルが多く裏方さんは疲れ切ってしまいましたが合宿自体は非常に楽しかったです。来年はもっと参加人数が増えるといいんですが、、、。次の裏方さんは頑張って下さい。影ながらもっとトラブルが起こることを祈っています、ちゃんちゃん。

一般記事

Direct3D Retained-Mode を full-screen で使う方法

1997 年 2 月 10 日

わたる

はじめに

Windows で 3D プログラミングを始めるに当たり、多くの人は DirectX3 SDK[1]の Help にある Direct3D Retained-Mode[2] Tutorial をまず読むでしょう。ところが、この Tutorial で紹介される Helworld.c は、DirectDrawClipper[3]を使った windowed mode のアプリケーションであり、やはり多くの人が望むと思われる full-screen mode ではありません。やむを得ず SDK Samples を解析することになるかと思いますが、full-screen の例は fly[4]のたった一つだけしかなく、しかもソースは多数のファイルに散らばっています。解析するのはひどくめんどうな作業となるでしょう。そこでこの記事では、Direct3D RM を full-screen mode に初期化する方法を、できるだけ簡潔に説明しようと思います。これを読んであなたの苦労が軽減されれば幸いです。

初期化の概要

初期化のおおまかな手順は以下の通りです。

1. 3D アクセラレーションを持つ DirectDraw オブジェクトを探す。
2. DirectDraw オブジェクトを生成し、画面を初期化する。
3. D3D HAL を探す。なければ MONO モデルの HEL を探す。
4. HAL(または HEL) の対応する bpp と z-buffer depth を調べる。

[1] http://www.microsoft.com/mediadev/から入手可能。

[2] Direct3D には Retained-Mode と Immediate-Mode があります。前者は OpenGL のように、3D CG を高度に抽象化したライブラリです。後者は三角形のポリゴンしか扱えませんが、ハードウェアを直接操作できる低レベルな API です。Retained-Mode は内部で Immediate-Mode を使っています。

[3] DirectDraw アプリの画面クリッピングを自動でやってくれるもの。ウインドウ内に描画するのに便利。DirectDraw はフルスクリーンでしか使えないと思っていたら誤解です。

[4] 飛行機が煙を吐きながら山の起伏にあわせて飛ぶデモ。

5. z-buffer を生成する。
6. Direct3D RM オブジェクトを生成する。
7. Direct3D RM Device オブジェクトを生成する。
8. レンダリングの品質を設定する。
9. Direct3D RM Viewport オブジェクトを生成する。

DirectDraw オブジェクトの探索

DirectDraw オブジェクトとは、要するにディスプレイドライバのことなのですが、Windows には複数の DirectDraw オブジェクトが存在し得ます。PowerVR や Voodoo のような、2D のビデオカードと併用するタイプの 3D アクセラレータを使っている場合が該当します。Direct3D を使うアプリケーションは、まず適切な DirectDraw オブジェクトを選ぶことから初期化を始めなければなりません。次の API を実行して下さい。

```
HRESULT DirectDrawEnumerate(LPDDENUMCALLBACK lpCallback,
                            LPVOID lpContext);
```

lpCallback で指定されたコールバック関数が、DirectDraw オブジェクトの GUID[5] を引数にして呼び出されます。もし DirectDraw オブジェクトが複数あれば、複数回呼び出されます。lpCallback は、以下のようなプロトタイプで、アプリケーションのプログラマが定義します。

```
BOOL WINAPI lpCallback(GUID FAR * lpGUID,
                       LPSTR lpDriverDescription,
                       LPSTR lpDriverName,
                       LPVOID lpContext);
```

ここの lpContext は、DirectDrawEnumerate() の lpContext に与えたものです。例えば次のように使います。

```
LPGUID lpDDGUID = NULL;
HRESULT rval = DirectDrawEnumerate(dd_enum_callback,
                                   &lpDDGUID);
```

こうすれば dd_enum_callback() の中で、lpDDGUID を書き換えらるようになります。DirectDrawEnumerate() の実行が終わった時点で、適切な DirectDraw オブジェクトの GUID が設定されているようにできるのです。なお、ここで GUID でなく LPGUID と宣言し、NULL に初期化するしているのには理由があり、後述します。

lpContext の他の使い方としては、C++で Direct3D をカプセル化する際に this を与えるという手があります。メソッドをコールバック関数にするには、static でなければなりませんが、

[5] Globally Unique IDentifier

this を与えておけば、static メソッドの中から、static でないメンバにアクセスできます。

lpCallback の定義に話を戻しましょう。このコールバック関数の中で、与えられた DirectDraw オブジェクトが、アプリケーションに相応しいかどうか判断します。具体的には、3D アクセラレーションがあるかどうか調べるのです。

```
BOOL WINAPI DirectDraw::dd_enum_callback(GUID *lpGUID,
                                         LPSTR lpDriverDesc,
                                         LPSTR lpDriverName,
                                         LPVOID lpContext)
{
    LPGUID *lplpDDGUID = (LPGUID *)lpContext;
```

lpContext に LPGUID へのポインタを与えた、先ほどの例の場合を見ていくことにしましょう。

```
    LPDIRECTDRAW lpTempDD;
    HRESULT rval = DirectDrawCreate(lpGUID, &lpTempDD, NULL);
    if(rval){
        Msg("不正な GUID が与えられました。");
        return DDENUMRET_OK;
    }
```

まず、与えられた GUID が本当に有効か試してみます。DirectDraw オブジェクトを生成するだけならば、画面になんら影響はありませんので、実際に生成してみるのが一番です。もし失敗したら、DDENUMRET_OK を戻り値として return すれば、列挙が続行されます。他にもまだ DirectDraw オブジェクトが残っていれば、再びコールバック関数が実行されるのです。列挙を中止してよい場合は、DDENUMRET_CANCEL を戻り値とします。

Msg() は SDK Samples の中で、ダイアログボックスを表示する関数として定義されています。しかし DirectDraw を用いたアプリは、画面を独占的に操作している関係上、GDI を使うダイアログボックスを正しく表示できないことが頻繁にあります。エラーメッセージはログファイルに残した方がよいでしょう。

```
HRESULT GetCaps(LPDDCAPS lpDriverCaps, LPDDCAPS lpHELCaps);
```

次に IDirectDraw::GetCaps を使って、生成した DirectDraw オブジェクトに 3D アクセラレーションがあるか調べます。GetCaps() は HAL[6] と HEL[7]の情報を一度に返すので、DDCAPS 構造体を二つ用意して下さい。

```
    DDCAPS HALCaps, HELCaps;
    ZeroMemory(&HALCaps, sizeof(DDCAPS));
    HALCaps.dwSize = sizeof(DDCAPS);
    ZeroMemory(&HELCaps, sizeof(DDCAPS));
    HELCaps.dwSize = sizeof(DDCAPS);
```

DDCAPS は必ずゼロクリアし、dwSize にその大きさを代入しなければなりません。IDirectDraw インターフェイスは COM[8]形式で提供されているので、将来バージョンアップする可能性があ

[6] Hardware-Abstraction Layer
[7] Hardware-Emulation Layer
[8] command.com の com ではなく、Component Object Model です。例によって Microsoft の新語ですが、クラス継承の概念そのまんまなのも毎度のことです。

ります。現に DirectX2 では、IDirectDraw2 インターフェイスに拡張されました。DDCAPS 構造体も、DirectDraw にあわせて変更されるかも知れません。DDCAPS のバージョンは dwSize によって判定されるのです。

```
rval = lpTempDD->GetCaps(&HALCaps, &HELCaps);
if(rval){
    Msg(rval,"DDCAPS の取得に失敗しました。");
    lpTempDD->Release();
    return DDENUMRET_OK;
}
if(HALCaps.dwCaps & DDCAPS_3D){
    //D3D HAL がある
    if(lpGUID){
        if(*lplpDDGUID == NULL)
            *lplpDDGUID = new GUID;
        memcpy(*lplpDDGUID, lpGUID, sizeof(GUID));
    }
}
else{
    //D3D HAL がない。
    /* DO NOTHING */
}
```

無事に HALCaps を取得できたら、dwCaps の DDCAPS_3D ビットがオンになっているか調べます。このビットが 3D アクセラレーションの有無を表しているのです[9]。もしオンになっていたら GUID をコピーします。ここで注意しなければならないことが二つあります。

一つ目は、lpGUID が NULL のときがあることです。NULL はデフォルトのディスプレイドライバを意味しており、DirectDrawCreate() は lpGUID に NULL が与えられると、現在使用しているディスプレイドライバに対応する DirectDraw オブジェクトを生成します。しかし使用中のディプレイドライバでも、律儀に正確な GUID を引き渡してくる場合はあるので、これをもとにドライバの状態を判定することはできません。

二つ目は、GUID とは ID なのだから実体は unsigned long だろう、などと勝手に決めつけてはならないことです。

```
typedef struct _GUID {
    unsigned long Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char Data4[8];
} GUID;
```

GUID は 128 ビット整数であり、C 言語では構造体として扱われるのです。

```
lpTempDD->Release();
if(*lpDDGUID){
    //D3D HAL を発見したので列挙中止
```

[9] メルコの ViRGE ドライバはオフになっています。Direct3D アプリが ViRGE の認識に失敗するようならば、SDK 付属の DirectDraw Cap Editor で強制的にオンにして下さい。

```
        return DDENUMRET_CANCEL;
    }
    //D3D HAL がまだ見つからないので列挙継続
    return DDENUMRET_OK;
}
```

3D アクセラレーションのある DirectDraw オブジェクトを発見できたら、列挙は中止して下さい。生成した DirectDraw オブジェクトを解放するを忘れずに。

DirectDraw オブジェクトの生成と画面の初期化

DirectDraw オブジェクトを選択したらさっそく生成します。DirectDraw2 オブジェクトもまとめて作ってしまいましょう。DirectDraw2 と Direct3D は、DirectX2 からサポートされました。だから DirectDraw2 が使えれば、DirectX2 ランタイムライブラリがインストールされていることを意味し、Direct3D も使えるはずです。

```
LPDIRECTDRAW lpDD1;
rval = DirectDrawCreate(lpDDGuid, &lpDD1, NULL);
if(rval){
    Msg("DirectDraw オブジェクトの生成に失敗しました。");
    exit(1);
}
LPDIRECTDRAW2 lpDD2;
rval = lpDD1->QueryInterface(IID_IDirectDraw2,
                             (void **)&lpDD2);
if(rval){
    Msg("DirectX2 以降をインストールして下さい。");
    exit(1);
}
else lpDD1->Release();
```

IID_IDirectDraw2 は DirectDraw2 の GUID です。dxguid.lib の中で定義されているのでリンクして下さい。#define INITGUID とすれば、dxguid.lib をリンクしなくても GUID を定義できますが、多重定義のトラブルが起きやすく、ライブラリをリンクする方が簡単かつ安全です。

```
rval = lpDD2->SetCooperativeLevel(hWnd,
                                  DDSCL_EXCLUSIVE  |
                                  DDSCL_FULLSCREEN |
                                  DDSCL_ALLOWREBOOT);
if(rval){
    Msg(rval,"Full-Screen モードにできません。");
    exit(1);
}
rval = lpDD2->SetDisplayMode(640, 480, 16, 0,0);
if(rval){
    Msg("画面解像度の変更に失敗しました。");
    exit(1);
}
```

Direct3D には 640x480 16bpp の画面モードが最適です。340x200 や 340x240 の、一般に MODE X と呼ばれているモードの方が、解像度が低くて速そうに思えますが、GDI のサポートしていない解像度なので、2D アクセラレーションの恩恵が受けられなくなります。また、これらの解像度をサポートしている 3D アクセラレータもあまりありません[10]。8bpp を使わないのもそのためです。

```
DDSURFACEDESC ddsd;
DDSCAPS ddscaps;
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    |
                      DDSCAPS_3DDEVICE;
ddsd.dwBackBufferCount = 1;
LPDIRECTDRAWSURFACE lpFrontBuffer, lpBackBuffer
rval = lpDD2->CreateSurface(&ddsd, &lpFrontBuffer, NULL);
if(rval){
    Msg("サーフェイスの生成に失敗しました。");
    exit(1);
}
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
rval = lpFrontBuffer->GetAttachedSurface(&ddscaps,
                                         &lpBackBuffer);
if(rval){
    Msg("BackBuffer の取得に失敗しました。");
    exit(1);
}
```

次にダブルバッファリング[11]できる primary surface[12]を生成します。必ず DDSCAPS_3DDEVICE を有効にして下さい。このビットがオンになっていないと 3D 描画には使えません。back buffer[13]へのポインタも取得しておいて下さい。後で必要になります。

D3D HAL の探索

いよいよ Direct3D オブジェクトを作る段階になりました。

```
LPDIRECT3D lpD3D = NULL;
//Direct3D オブジェクトを生成。
rval = lpDD2->QueryInterface(IID_IDirect3D,
                             (void**)&lpD3D);
```

[10] ViRGE は 340x200、340x240、512x384、640x400 などが使えます。
[11] 画面をふたつ用意して、表示用と作画用に分けること。書き換え中の画面を見せなくできます。
[12] 表示されてい画面のこと。複数の overlay surface によって構成され、各 overlay surface は front buffer と back buffer からなります。overlay 表示をしない場合は primary surface = front buffer です。overlay 表示する場合でも、ソフトウェアでエミュレートする場合は事情が異なります。
[13] 作画用の画面のこと。flip すると front buffer と入れ替わります。

```
    if(rval){
        Msg(rval,"Direct3D を使用できません。");
        exit(1);
    }
```

Direct3D オブジェクトは DirectDraw オブジェクトから召還します。Direct3D オブジェクトは、光源や物体の材質など、仮想三次元空間の物理的性質を調整するものです。画面表示をどれくらい美しくするかという、レンダリングの品質を設定をするのは、Direct3D Device オブジェクトです。レンダリングを CPU と 3D アクセラレータのどちらが行うのかは、どちらの Direct3D Device を選ぶかによって決まるのです。3D アクセラレータを持つ DirectDraw オブジェクトを生成すれば、即ハードウェアによる 3D 描画ができるのではありません。DirectDraw の選択は、Direct3D Device の選択肢の範囲を調節したにすぎないのです。どんな Direct3D Device オブジェクトを使用できるかは、IDirect3D::EnumDevices によって調べられます。

```
HRESULT EnumDevices(LPD3DENUMDEVICESCALLBACK lpCallback,
                    LPVOID lpArg);
```

DirectDrawEnumerate() とそっくりな形をしています。使い方も同じです。

```
  GUID D3DevGUID;
  rval = lpD3D->EnumDevices(enum_device_func, &D3DDevGUID);
  if(rval){
      Msg("Direct3D Device の列挙に失敗しました。");
      exit(1);
  }
```

DirectDraw のときと違うのは、コールバック関数のプロトタイプです。以下のように大変複雑な形をしています。

```
typedef HRESULT (FAR PASCAL * LPD3DENUMDEVICESCALLBACK)
                (LPGUID lpGuid,
                 LPSTR lpDeviceDescription,
                 LPSTR lpDeviceName,
                 LPD3DDEVICEDESC lpD3DHWDeviceDesc,
                 LPD3DDEVICEDESC lpD3DHELDeviceDesc,
                 LPVOID lpArg);
```

引数がたくさんあるのは、調べなくてはならないことがたくさんあるためです。

```
HRESULT WINAPI enum_device_func(LPGUID lpGuid,
                                LPSTR lpDeviceDescription,
                                LPSTR lpDeviceName,
                                LPD3DDEVICEDESC lpHWDesc,
                                LPD3DDEVICEDESC lpHELDesc,
                                LPVOID lpContext)
{
    LPGUID lpD3DDevGUID = lpContext;
    LPD3DDEVICEDESC lpDesc;
```

```
lpDesc  = lpHWDesc->dcmColorModel ? lpHWDesc : lpHELDesc;
```

まず列挙された Direct3D Device オブジェクトが、ソフトウェアエミュレーションのものなのか、3D アクセラレータのものなのか調べます。もし 3D アクセラレータならば、lpHWDesc->dcmColorModel が 0 以外の値になっているので、これをもとに判別して下さい。

```
if( !(lpDesc->dwDeviceZBufferBitDepth & DDBD_16) )
    return D3DENUMRET_OK;
```

次に z-buffer の精度を調べます。16bpp が適当でしょう。8bpp だと精度が足らず、24bpp や 32bpp だとメモリを食いすぎます。3D アクセラレータを使用する場合は、z-buffer や texture を、ビデオメモリ上に作らなければならないので、メモリの消費量には特に気をつける必要があるのです。16bpp が使用できない Direct3D Device なら列挙を続行し、次のオブジェクトに移って下さい。

```
if( !(lpDesc->dpcTriCaps.dwTextureCaps) )
    return D3DENUMRET_OK;
```

次は texture-mapping が表示できるのかどうか調べます。こんなことをわざわざ調べる必要性があるのか、疑問に思う人もいるでしょう。しかし絶対に必要なことなのです。あまり知られていないことですが、高速なビデオカードとして有名な MGA Millennium は、CAD 向けに 3D ポリゴン描画機能を持っているのです。Direct3D でもその機能を使えるのですが、Millennium のポリゴンができるのはグローシェーディングまでで、残念なことに texture-mapping は表示できないのです。

```
if( !(lpDesc->dwDeviceRenderBitDepth & DDBD_16) )
    return D3DENUMRET_OK;
```

今度は 16bpp の画面に対応しているか調べます。8bpp や 24bpp、32bpp に描画できない 3D アクセラレータはありますが、16bpp が使えないアクセラレータは現時点ではありません。しかし一応確かめておくべきでしょう。

```
if(lpDesc == lpHWDesc){
    //D3D HAL ならば列挙中止
    bIsD3DHAL = TRUE;
    memcpy(lpD3DevGUID, lpGuid, sizeof(GUID));
    return D3DENUMRET_CANCEL;
}
else{
    //HEL ならば列挙続行
    bIsD3DHAL = FALSE;
    if(lpDesc->dcmColorModel == D3DCOLOR_MONO){
        memcpy(lpDevGuid, lpGuid, sizeof(GUID));
    }
}
return D3DENUMRET_OK;
```

ここまでの検査に合格した Direct3D Device オブジェクトが、3D アクセラレータのならば、文句なく使用できます。GUID をコピーして下さい。それから、3D アクセラレータであることを示すフラグを作って下さい。後で z-buffer を生成するときに必要となります。

もしソフトウェアエミュレーションならば、最後のチェックが残っています。Color Model[14]を調べて下さい。MONO モデルならば、エミュレーションでも実用的な速度で描画できるので

[14]光の扱い方のこと。RGB モデルは光の色をレンダリングに反映しますが、MONO モデルは光の明るさしか考慮されません。

すが、RGB モデルは MMX を搭載していない限り、非常に遅くなってしまうのです。また、列挙を続行すれば 3D アクセラレータが見つかる可能性があります。

z-buffer の生成

z-bufferは、Retained-Modeの初期化の前に用意しなければなりません。Direct3D RM Device オブジェクトは、生成されたときにだけ z-buffer の有無を調べるからです。ちなみに、z-buffer がない場合は z-sort 法で描画されますが、Direct3D RM Face[15]単位でソートするのではなく、Direct3D RM Frame[16]単位のソートなのです。Face は登録された順番で描画されてしまうので、上下関係が簡単に破綻してしまいます。さらに z-sort だと texture-mapping のパフォーマンスも悪化します[17]。Direct3D で z-sort を使うのは現実的とは言えません。

```
DDSURFACEDESC ddsd;
ZeroMemory(&ddsd, sizeof(DDSURFACEDESC));
ddsd.dwSize         = sizeof(DDSURFACEDESC);
ddsd.dwFlags        = DDSD_WIDTH  |
                      DDSD_HEIGHT |
                      DDSD_CAPS   |
                      DDSD_ZBUFFERBITDEPTH;
ddsd.dwWidth        = 640;
ddsd.dwHeight       = 480;
ddsd.ddsCaps.dwCaps = DDSCAPS_ZBUFFER;
if(bIsD3DHAL)
    ddsd.ddsCaps.dwCaps |= DDSCAPS_VIDEOMEMORY;
else
    ddsd.ddsCaps.dwCaps |= DDSCAPS_SYSTEMMEMORY;
ddsd.dwZBufferBitDepth = 16;
LPDIRECTDRAWSURFACE lpZBuffer;
rval = lpDD2->CreateSurface(&ddsd, &lpZBuffer, NULL);
if(rval){
    Msg("z-buffer の生成に失敗しました。");
    exit(1);
}
```

3D アクセラレータの場合、z-buffer は必ずビデオメモリ上に作らなければなりません。反対に、ソフトウェアエミュレーションの場合、CPU 側のメモリ上に作る必要があります。`CreateSurface()` は、生成先のメモリが明示的に指定されていないと、まずビデオメモリを確保しようとし、失敗すればシステムメモリを確保しようとします。つまり、メモリの種別を指定しなくても、たまたま動いてしまうことがあり得るのです。くれぐれも忘れないように気をつけて下さい。

```
rval = lpBackBuffer->AddAttachedSurface(lpZBuffer);
if(rval){
    Msg("z-buffer を Back-Buffer に接続できません。");
    exit(1);
```

[15] Direct3D RM ではポリゴンのことを face と言います。
[16] ここでは三次元上の物体であると理解していれば十分です。
[17] 同じ texture のポリゴンをまとめて描くと、texture がキャッシュに載るので速くなります。z-sort では原理的に不可能です。

```
}
```

ダブルバッファリングなのでレンダリングは back buffer にします。z-buffer は back buffer にだけ必要です。

Direct3D RM の作成と Direct3D RM Device の生成

Direct3D RM オブジェクトを作るには何の準備もいりません。

```
LPDIRECT3DRM lpRM;
rval = Direct3DRMCreate(&lpRM);
if(rval){
    Msg("D3D Retaind-Mode を使用できません。");
    exit(1);
}
```

一方、Direct3D RM Device オブジェクトを生成するには、先に Direct3D Device オブジェクトを生成する必要があります。

```
LPDIRECT3DDEVICE lpD3DDevice;
rval = lpBackBuffer->QueryInterface(D3DevGUID,
                                    (void**)&lpD3DDevice);
if(rval){
    Msg(rval,"D3D Device の生成に失敗しました。");
    exit(1);
}
```

ここで先ほど得た GUID が必要になります。Direct3D Device オブジェクトは、レンダリング先の DirectDrawSurface から作らなければなりません。一般に DirectX において、ある COM から別の COM を QueryInterface() できるのは、メモリ上の実体が同じ COM に限ります。メモリをコピーして別実体の COM を作るには、Direct3D RM の場合、IDirect3DRMObject::Clone を使います。

```
LPDIRECT3DRMDEVICE lpRMDevice;
rval = lpRM->CreateDeviceFromD3D(lpD3D, lpD3DDevice,
                                 &lpRMDevice);
if(rval){
    Msg("D3DRM Device の生成に失敗しました。");
    exit(1);
}
rval = lpRMDevice->SetBufferCount(2);
if(rval){
    Msg("D3DRM Device の設定に失敗しました。");
    exit(1);
}
lpD3D->Release();
lpD3DDevice->Release();
```

Direct3D RM Device オブジェクトの作成はなかなか複雑です。Direct3D RM オブジェクトから、Direct3D オブジェクトと、Direct3D Device オブジェクトを引数として、作成するのです。作成できたら IDirect3DRMDevice::SetBufferCount を実行して下さい。ダブルバッファリ

ングをする場合は 2 を、トリプルバッファリング[18] する場合は 3 を、バッファリングしない場合は 1 を指定します。これは Direct3D RM が、画面の差分描画を行うために必要な設定です。レンダリングは大変重い処理なので、画面の実際に変わる部分だけを再描画するわけです。しかしそのためには、前回、前々回、あるいは前々々回のレンダリングでどこに何を描いたか覚えていなければなりません。Retained-Mode はこの辺りを自動的にやってくれるので大変楽です。もし、Immediate-Mode で Retained-Mode のようなライブラリを作ろうとしたら、どんなに途方もない作業になるか、想像もつきません。

ここまでの作業が終われば、Direct3D オブジェクトと Direct3D Device オブジェクトは解放して構いません。

レンダリングの品質の設定

前の方でも書きましたが、Direct3D Retained-Mode で、レンダリングの品質を制御しているのは、Direct3D RM Device オブジェクトです。ポリゴンごとに品質を変更することはできますが、Direct3D RM Device オブジェクトの設定がデフォルトとなるのです。

```
rval = lpRM->SetDefaultTextureColors(48);
rval = lpRM->SetDefaultTextureShades(16);
rval = lpRMDevice->SetShades(8);
rval = lpRMDevice->SetQuality(D3DRMRENDER_GOURAUD);
rval = lpRMDevice->SetDither(FALSE);
D3DRMTEXTUREQUALITY quality;
if(bIsD3DHAL)
    quality = D3DRMTEXTURE_LINEAR;
else
    quality = D3DRMTEXTURE_NEAREST;
rval = lpRMDevice->SetTextureQuality(quality);
```

この設定内容は、16bpp のときの最も一般的と言えるものですが、アプリケーションの都合にあわせて変更できます。上の例では、3D アクセラレータが使える場合にだけ、bi-linear filtering[19]をするようにしています。ソフトウェアエミュレーションで使うには、処理が重すぎるからです。

Direct3D RM Viewport を生成

Direct3D は、device と viewport が分けて管理されています。Device に複数の Viewport を作ることができるからです。例えばレーシングゲームで、バックミラーやサイドミラーに後方

[18]ダブルバッファリングは、画面切り替えのために V-Sync を待つ時間が無駄になります。トリプルバッファリングなら待っている間も描画できるので、パフォーマンスがよくなります。ビデオメモリを馬鹿食いするのが難点です。

[19]アンチエイリアス技術の一つ。ポリゴンが画面に拡大されると、texture のピクセルがタイルのよう大きく見えてしまいます。これをぼかしてごまかします。効果絶大です。

の様子を写すのに使えます。Direct3D RM Viewport オブジェクトを作るには、先にシーンフレーム[20]とカメラフレームが必要です。

```
LPDIRECT3DRMFRAME lpScene, lpCamera;
rval = lpRM->CreateFrame(NULL, &lpScene);
rval = lpRM->CreateFrame(lpScene, &lpCamera);
```

lpCamera は lpScene のヒエラルキー上にさえあればよく、直接の子供である必要はありません。

```
LPDIRECT3DRMVIEWPORT lpRMView;
rval = lpRM->CreateViewport(lpRMDevice,
                            lpCamera,
                            0, 0,
                            640, 480,
                            &lpRMView);
rval = lpRMView->SetBack(D3DVAL(5000.0));
```

Direct3D RM Viewport オブジェクトを生成したら、後方クリッピングの距離を設定して下さい。

終わりに

画面をレンダリングするためには、まだ光源の設定と物体の配置が必要ですが、Direct3D Retained-Mode の初期化は以上で終わりました。

DirectX がハードウェアの直接操作をコンセプトとしているため、めんどうな手続きが延々とありました。しかし、これらは Windows アプリの多くの処理と同様に定型的で、ライブラリ化が容易です。次の部報オリパンフ号では、DirectX を C++でカプセル化するために、私が作ったクラスライブラリ TGL(仮称)[21]を紹介したいと思います。

Direct3D は Microsoft が作ったのではなく、Reality Lab. を買い取ったものです。おかげでバグもなくよくできており、なかなか使いやすいです。OpenGL のような汎用性こそありませんが、リアルタイム 3D 処理に興味のある人には、自信を持って勧められるものに仕上がっていると思います。

> Direct3D の原稿は pLaTeX2e で書きました。しかし、
> 『IDirect3DRMDevice::SetBufferCount』など、長い単語がぼこぼこあるせいで、TeX はちょうどよい改行位置を見つけられず、エラーが出まくりです。結局整形は編集長にまかせてしまいました。TeX がコンピュータの記事に弱いなんて意外です。C 言語のソースを自動整形できたらいいのに。

[20]フレームとは Retained-Mode の用語で座標系のこと。シーンフレームが絶対座標系で、他のフレームの座標はシーンフレームに対して相対的なもの。フレームはツリー構造のヒエラルキーを作れるので、多関節物体を容易に設計できます。

[21]http://www.komba.ecc.u-tokyo.ac.jp/~g540879/d3d/repository.html で公開中。

ホームページ上で掲示板を作ろう !!

ばんだい

ホームページはどういうもの？

普通に文字だけを書いたホームページというのはそのページを見てる人に一方的に情報を伝えることしか出来ません。いろいろな情報を見るというのはそれだけでも楽しいものです。しかし、ホームページをもっと面白くする方法があります。それはいわゆる「双方向化[1]」をすることです。

つまり、ページを見ている相手からもこちらが情報を受け取れるようにするのです。そうすれば作っている側としても、自分のページについての反響などを知ることが出来て嬉しいものです。今回はそれを実現する一つの方法として cgi を使った一行掲示板の作り方を紹介しましょう。

掲示板を実現する方法はいろいろありますが、これから紹介する方法は私が実際に使っている方法です。この方法は必ずしもいいものとは言えないかも知れませんが、とりあえずこれでちゃんと動いているので紹介します。

今回実際に作るのは、ホームページ上で入力されたものを単にファイルに書き込むと言うものです。

フォーム

フォームを示すタグ

ホームページ上でそれを見ている人からの入力を受け取るには html の<form>というタグを使います。この書式は次の通りです。

```
<form method=受渡し方法 action=処理プログラム>
.....
</form>
```

[1] こう言うとなんかえらそうですね

method

method に指定する受渡し方法には POST と GET があります。動作はそれぞれ次の表のようになっています。

GET	受け取ったデータを環境変数で処理プログラムに渡す
POST	受け取ったデータを処理プログラムの標準入力に渡す

環境変数の領域はサイズが決まっているらしいので、標準入力に渡したほうが多くのデータを渡せるそうです。しかし、POST を使うと、データを送る前にいちいちセキュリテリーがどうのという確認のダイアログがでてめんどくさいので、そういうのが出ない GET を使います。

action

action には渡したデータを実際に処理するプログラム名を入れます。普通は拡張子が cgi のものを入れます。

入力領域

次に実際にデータを入力出来る領域を作ります。そのためには次のような命令を使います。

１行テキスト入力

まずは１行のテキストを入力できる領域を作る方法です。

```
<input name=名前 size=大きさ>
```

こう書くと幅が size の１行テキスト入力領域ができます。ここに書いたことは name とともに処理プログラムに渡されます。name はたいして重要ではありませんが変数名だと思ってくれれば良いです。

提出ボタン

```
<input type="submit" value=ボタンの文字>
```

これはさっき出て来た input ですが、type という項目が付いています。こうすると、value で指定した文字が書かれたボタンが出来て、このボタンを押すと、さっきの form タグの action で指定したプログラムに入力した内容が渡されます。

やりなおし

```
<input type="reset" value=ボタンの文字>
```

これも前のと type の値が違うだけですが、これも value で指定した文字が書かれたボタンが出来ます。しかし、今度はこれを押すと、テキスト入力領域に入力した文字が消えます。つまり、書き直しが出来るのです。

実例

ではこれらをつかって実際に入力フォームを作ってみましょう。これには keiji.html という名前を付けておきます。

```
<form method="GET" action="keiji.cgi">
名前：<input name="name" size=20><br>
ひとこと：<input name="text" size=80><br>
<input type="submit" value="送る">
<input type="reset" value="やり直し"><br>
</form>
```

これで入力のページは完成しました。テキスト入力領域に文を書いて「送る」のボタンを押せば keiji.cgi にその文がわたされることになります。

どのようにして処理をするか

ここまでできて、次に必要なのは action に指定する受け取ったデータを処理するプログラムです。ここではそのプログラムを作るのに sh シェルスクリプトを使います。シェルスクリプトとは MS-DOS のバッチファイルのようなもので、手軽に書けるのが利点です。変数の定義や代入が簡単で if 〜 elif などの制御文もそなえているので非常に便利です。

sh

ここでは掲示板を作るのに必要なコマンドだけ紹介します。

echo とリダイレクト

```
echo 文字列>>ファイル名
```

言わずと知れた echo とリダイレクトです。受け取ったデータをファイルに保存する一番簡単なやりかたがこうです。

変数代入と参照

```
変数名=文字列
${変数名}
$変数名
```

このように書くだけで変数に文字列が入ります。この手軽さが sh の良いところです。この変数の値が欲しいときは${変数名}と書くだけで参照できます。$変数名のように、{}は無くてもかまいません。

標準出力取り込み

‘コマンド‘

コマンドを ‘‘(バッククオート) でかこむと、その結果出力されるものが書いた場所に入ります。これはどんなときに使えるかと言うと、あるコマンドの出力を別のコマンドの引数に渡したり、変数に格納したりするときです。つまり、ちょっと極端な例ですが、

```
var=‘ls‘
```

等とやると ls の出力がそのまま var に入ったりします。

どのように渡されるか

QUERY_STRING

先に書いたようにこのフォームでは「送る」のボタンを押せば文がプログラムに送られますが、これはどのようにわたされるのでしょうか。method に GET を指定した場合は環境変数に入ると書きましたが、具体的には

```
QUERY_STRING
```

という変数に入ります。だから受け取ったデータは

```
$QUERY_STRING
```

で参照できるのです。では、さっきのフォームに入力をするとどのように渡されるのでしょうか？例えば、「名前」に bandai、「ひとこと」に hogehoge と入れて、「送る」ボタンを押すと、QUERY_STRING には次のようなものが入ります。

```
name=bandai&text=hogehoge
```

ここで、name text はそれぞれ入力領域に付けられた変数名でしたね。つまり、

```
変数名=入力された文字列&変数名=文字列....
```

というものが入ります[2]。

文字列の切り出し

さて、今作っている掲示板のプログラムは入力された文をファイルに書き込まなくてはいけません。書き込むものは実際に入力されたものだけですから、QUERY_STRING に入ったものには邪魔なものがたくさん付いています。そこで、ここから入力された文字列だけを切り出さなくてはなりません。しかし、この作業はたぶん sh シェルスクリプトだけではできません[3]。そこで、ここではその処理の部分だけを外部プログラムとして C で作って、スクリプト内から呼び出すことにします[4]。

[2] この形式のものが URL にもくっつきます
[3] Perl だとできるのになぁ…
[4] この方法だと遅いかも知れません

では、その切り出しプログラムはどのように作れば良いでしょうか？さっきの QUERY_STRING の例を見れば、=から&までの間の文字列を取り出せばよいことがわかります。ここではこの処理をするプログラムを separete とし、次のような仕様とします。

separete 文字列 数字
「文字列」(QUERY_STRING の中の文字列) の「数字」番目のパラメータを切り出し、それを標準出力に出力する。

このようにすると、

```
param1=‘separate $QUERY_STRING 1‘
```

とするだけで 1 つめの入力領域に入力された文字列が変数 param1 に取り出せることになります。

デコーディング

入力された文字列が英数字だけならこれだけでいいのですが、渡すデータに日本語が入っているとちょっと厄介になります。日本語はエンコードされてここに入るからです。エンコードの方法は簡単で、

%16 進文字コード

と言う形式です。また、半角のスペースも+に置き換わります。だからこれらをデコードしてやらなくてはなりません。しかしこれも sh でするのはたぶん無理です。そこで、ここも外部プログラムに頼ることになります。このプログラムを decode とし、仕様を次のようにします。

decode 文字列
「文字列」をデコードして、標準出力に出力する。

こうするとこれも、

```
text2=‘decode $text‘
```

で text2 に入力されたものと同じ文字列が入ることになります。

書き込み

ここまできたらあとはファイルに書き込むだけです。書き込むのは簡単で、リダイレクトを使うだけです。では、文字列を受け取って、ファイルに書き込む cgi プログラム keiji.cgi を作ってみましょう。

```
#!/bin/sh
param=$QUERY_STRING
name=‘separate $param 1‘
text=‘separate $param 2‘
name2=‘decode $name‘
text2=‘decode $text‘
echo $name2’<br>’$text2’<hr>’>>keijilog.html
```

これで、フォームに入力された文字列が keijilog.html に書き込まれます。

画面への出力

これで受け取った文字列をファイルに書き込むことはできましたが、それだけではいけません。このプログラムはブラウザから呼ばれているので、このプログラムも何かを表示しなくてはいけません。表示をしないとエラーになってしまいます。ところで、cgi から html を出力するときには大事なことがあります。

```
Content-type: text/html
(改行)
```

という文を必ず最初に出力しなければなりません[5]。こうしなければたぶんエラーになります。では、実際にこの部分を作りましょう。さっきの keiji.cgi の後に次のように付け加えます。ここでは、一言と、掲示板へ戻るリンク、掲示板を見るリンクを画面に出力します。

```
echo 'Content-type: text/html'
echo
echo '<font size=5>'
echo $name2'さん、どうもありがとうございました。'
echo '</font><br>'
echo '<a href="keiji.html">掲示板に戻る</a><br>'
echo '<a href="keijilog.html">掲示板を見る</a>'
```

さらなるバージョンアップへ

これで一通りちゃんと書き込めるホームページが出来上がりました。しかし、これは本当に一番簡単なもので、もっと直すべきところもありますし、もっと工夫を凝らしたものにすることも出来ます。私が作ったこれのもう少しちゃんとしたものが私のホームページにありますので、見ていただけると幸いです。URL は

http://www.komaba.ecc.u-tokyo.ac.jp/~g640834/24net/linex/linex3.cgi

です。

しかし、やっぱりシェルスクリプトで書くのはきついので、今度から Perl で書きたいと思います。Perl の方が機能も多いですからね。というわけで、次回から Perl を使った cgi 入門です。

作用の周辺

うえ

[5] MIME とかいうものでしょう

始めに

これは今年始まる「理論科学分科会」の原稿第 0 号ということにします。コンピュータとはなんら関係がないですがまあいいでしょう。

物理をやっていると結構いろいろなところで「作用」という量およびその方程式が顔を出します。そこでこの「作用」をテーマにいろいろうだうだと書いてみました。適当に読み流して下さい。

解析力学概論

この章では解析力学の結果および作用についての考察をできるだけ手短にまとめていますが、いかんせんこれだけでは始めて解析力学に触れる人にはわからないと思いますので、そのへんは各自で補完して下さい[1]。

最小作用の原理

力学法則の最も一般的な定式化は「最小作用の原理」によって与えられます。この原理によると、力学系は「ラグランジアン」$L = L(q, \dot{q}, t)$ なる量で特徴づけられ、系は「作用」という量 $S = \int_{t_1}^{t_2} L(q_i, \dot{q}_i, t)dt$ を可能な限り最小にするように運動します。

この条件は $\delta S = 0$ という形で書け、(δ は微小変分を表す) これだけの条件により非常に強力な「ラグランジュ運動方程式」

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} = \frac{\partial L}{\partial q_i}$$

が導かれますが、その過程は省略。ちなみに、「ラグランジアン」L はいろんな考察により結局「(運動エネルギー)−(ポテンシャルエネルギー)」で表されることがわかります。

また、この形式を用いると、

- 時間の一様性により、エネルギー保存則が
- 空間の一様性により、運動量保存則が
- 空間の等方性により、角運動量保存則が
- 時間の等方性により、力学法則の可逆性が

それぞれ導かれ、非常に美しい理論体系をなしています。

[1] 参考図書 ··· ランダウ、リフシッツ「力学」

正準方程式

以上で述べたラグランジュ方程式は一度使ったことのある人ならわかるでしょうが、非常に実際の問題を解くのに有用です。しかしこれから述べる方程式 (特にハミルトン－ヤコビ方程式) は、はっきりいって現在実用上我々が使うには面倒過ぎます。しかし、力学以外のところで結構顔を出すので知っておいて損はありません。

位置と座標の関数としてのラグランジアンの完全微分は

$$dL = \sum_i \frac{\partial L}{\partial q_i} dq_i + \sum_i \frac{\partial L}{\partial \dot{q}_i} \dot{dq}_i$$

で表されます。これをラグランジュ方程式を使ったりしてあれこれと変形していけば、結局「ハミルトンの正準方程式」

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \dot{p}_i = -\frac{\partial H}{\partial q_i}$$

が導かれます。ラグランジュ方程式 (およびニュートン方程式) が 2 階の微分方程式であったのに対し、ハミルトン方程式は 1 階の微分方程式であることが大きな特徴となっています。(その代わり式の数が 2 倍になった) ここで登場した H という量は「ハミルトニアン」と呼ばれており、$H(p,q,t) = \sum_i p_i \dot{q}_i - L$ と定義される代物です。少し計算すればわかりますが、エネルギーが時間に陽によらないときはハミルトニアンはエネルギーと同じになります。

この方程式は「位相空間」、すなわち与えられた力学系の s 個の一般座標と s 個の運動量の値を座標とする 2s 次元の空間における運動の解析に使ったりしますが、面倒ですし今回のテーマである「作用」をあらわには含まないので省略。

ハミルトン－ヤコビの方程式

さて、今度の法則は「作用」が主役です。

が、非常にわけわかんない方程式になっています。はっきりいって実用性は薄いでしょう[2]。

この方程式を導く前に、「作用」についてもう少し考察してみましょう。作用とは

$$S = \int_{t_1}^{t_2} L dt$$

のことでした。力学系はこの値を最小にするように動くわけですが、ここで作用の概念を別の形で考えましょう。つまり、S は本当の軌跡に沿った運動を特徴づける量であるとみなしましょう。つまり、作用積分を、積分の上限に相当する時刻における座標の値の関数として考えるわけです。

この考えにより、作用の座標に関する変分をとれば結局

$$\delta S = \sum_i p_i \delta q_i$$

[2]かつては「天体力学の奥義」だったそうですが

となりますので、結局

$$\frac{\partial S}{\partial q_i} = p_i$$

という重要な関係式が導けます。また、作用の定義そのものにより、

$$\frac{dS}{dt} = L$$

一方、S をここまでに述べた意味で座標と位置の関数とみなしておくと

$$\frac{dS}{dt} = \frac{\partial S}{\partial t} + \sum_i p_i \dot{q}_i$$

従って変形すると

$$\frac{\partial S}{\partial t} = L - \sum_i p_i \dot{q}_i = -H$$

となり、結局、重要な等式

$$\frac{\partial S}{\partial t} = -H$$

が導かれます。

さて、以上でハミルトン－ヤコビ方程式を導く準備は整いました。座標と時間の関数としての「作用」は、ハミルトニアンと

$$\frac{\partial S}{\partial t} + H(p, q, t) = 0$$

なる関係を持っていますが、ここで、一般化運動量 $p_i = \frac{\partial S}{\partial q_i}$ だったのだから結局関数 $S(q, t)$ が満たすべき方程式

$$\frac{\partial S}{\partial t} + H(q_1, \cdots, q_s; \frac{\partial S}{\partial q_1}, \cdots, \frac{\partial S}{\partial q_s}; t) = 0$$

が導かれます。これをハミルトン－ヤコビの方程式と呼びます。

この式を頑張って解くと、(うまくいくと) 自由度が s の系においては任意定数を $2s$ 個含む解が出てきて、非常に強力な結果が得られます。ただ、非常に面倒ですし[3]はっきりいってこんなことやらなくてもラグランジュ方程式さえあれば大抵のことはうまくいくので解き方をマスターする必要はないでしょう[4]。ただ、この方程式の知識は物理のさまざまな領域で有用です。その例を以下に示します。

[3] しかも難解
[4] 多分

前期量子論へ

波動関数と作用

作用の満たす方程式に

$$\frac{\partial S}{\partial x} = p, \frac{\partial S}{\partial t} = -H$$

がありましたから、エネルギーが保存される場合にはエネルギーを E として

$$s = \int \sqrt{2m(E - U(x))}dx - Et$$

が成立します。(任意定数は無視)[5]

波動関数について考えてみましょう。de Broglie の立てた仮説によると、自由粒子の場合、作用関数を $\hbar$ で割ったものが波の位相になっています。つまり、この時波動関数は

$$e^{i\frac{S(x,t)}{\hbar}}$$

と表されるわけです。一般の場合にも同じ形式で表されると考えてみましょう。プランク定数が十分に小さい ($\hbar \to 0$) とすると、この波は位置や時間の変化とともに激しく振動します。言い換えるとこの波の波数や振動数は十分に大きいわけです。粒子のエネルギーが正確には定まっていなくて、ある微小な幅でしか分からないとしてみましょう。さらに、このような場合、対応する波は正確なエネルギーを持った波がその幅にわたって重ね合わされたものであると仮定してみましょう。エネルギーが微小に異なる波が重ね合わされると、互いの位相が少しでも違うところでは振動のため波は打ち消し合う一方、位相が互いに等しいところでは二つの波が強め合います。このようにしてできた波を波束とよびます。エネルギーの変化があっても位相が変化しない位置が波束の位置になるため、波束の位置は作用関数を E で微分してゼロとおいて得られる次式を満たします。

$$0 = \frac{dS(x,t)}{dE} = -t + \int \sqrt{\frac{m}{2(E - U(x))}}dx$$

これをさらに時間で微分すると

$$\frac{dx}{dt} = \sqrt{\frac{2(E - U(x))}{m}}$$

となり、波束の運動速度が得られます。この式はまさしく運動方程式から導き出される粒子の速度なのです。つまり粒子の運動は作用関数を位相とする波を考えた時に干渉により波が強めあってできた波束の運動にちょうど一致しています。これから、十分に振動数が高くしかも波長の短い波が適当に重ね合わされたときの波束が粒子であると言う解釈が成り立つわけです。

[5] もちろん $\sqrt{2m(E - U(x))} = p$

また、「角運動量の量子化」の理論もこの形式から導き出されます。系がある周期運動を行なって元に戻った際には当然波の位相は元と同等であるべきですから、

$$\oint p(x)dx = 2n\pi\hbar = nh(n = 1, 2, \cdots)$$

が導けます。これを「Bohr-Sommerfelt[6]の量子化の条件」と言います。

演算子の正当化

さて、量子力学においては物理量は演算子で表現されます。

$$\hat{p} = -i\hbar\frac{\partial}{\partial x}, \hat{H} = i\hbar\frac{\partial}{\partial t}$$

となりますが、このことは量子化学や初等的な量子論ではもはや前提として与えられていますが、作用を用いた理論ではこのことを正当化できます[7]。実際に計算してみましょう。

波動関数を $\psi = e^{i\frac{S}{\hbar}}$ とおけることは前の節で述べました。実際にこの関数に微分演算子を作用させてみましょう。

$$\hat{p}\psi = \frac{\partial S}{\partial x}\psi$$

ですが、2 章で述べたとうり $\frac{\partial S}{\partial x} = p$ ですから、

$$\hat{p}\psi = p\psi$$

従ってこの表現は正しいことになります。

$\hat{H}$ についても同じことが言えます。計算してみて下さい。

以上のような理論からもわかるように、一見役に立たないようにみえる「作用」にも結構活躍する機会はあるわけです。

相対論的ハミルトン－ヤコビ方程式

相対論においてもハミルトン－ヤコビ方程式に相当するものはあります。しかし (特殊) 相対論について基礎から書き下していくとこの文章が終るわけもないのでまたしても概略だけ書くこととします[8]。

相対論的力学においても $p_i = \frac{\partial S}{\partial x^i}$ が成り立ちます。ここで X の添字が上にきていますが気にしないで下さい。ちなみに相対論ですから座標は 3 つではなく、4 つです。(ct, x, y, z) また、この (4 元) 運動量の間には

$$g^{ik}p_i p_k = m^2c^2$$

[6]つづりはこれで良かったっけ？
[7]別の方法もあります。「現代の量子力学」参照
[8]ここの文章は相対論の知識がないときつい

という等式が成立します。ちなみに相対論における慣例に従って同じ符合については和をとることになっています。ここで、g^{ik} は計量テンソルであり、この成分を行列の形で書くと

$$g^{ik} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

となります。

これを使って４元運動量についての関係式において和をあらわに書くと

$$\frac{1}{c^2}(\frac{\partial S}{\partial t})^2 - (\frac{\partial S}{\partial x})^2 - (\frac{\partial S}{\partial y})^2 - (\frac{\partial S}{\partial z})^2 = m^2c^2$$

が得られます。これが相対論的力学におけるハミルトン－ヤコビ方程式です。

この方程式の古典力学への移行を行なうためには相対論で出てくる物体の「静止エネルギー」mc^2 を考慮しなくてはなりません。作用 S はエネルギーと $E = -\frac{\partial S}{\partial t}$ によって結びつけられていますから、古典力学へ移行するには $S = S_1 - mc^2t$ によって新しい S_1 に置き換えなくてはなりません。これを与えられた式に代入すると

$$\frac{1}{2mc^2}(\frac{\partial S_1}{\partial t})^2 - \frac{\partial S_1}{\partial t} - \frac{1}{2m}[(\frac{\partial S_1}{\partial x})^2 + (\frac{\partial S_1}{\partial y})^2 + (\frac{\partial S_1}{\partial z})^2] = 0$$

となります。$c \to \infty$ なる極限移行を行なえば、これは古典論に出てくるハミルトン－ヤコビ方程式となります[9]。

次回予告

まだまだ未熟なもので「作用」が顔を出す分野は以上にあげたような分野と電磁気学の変分原理による定式化の部分しか知りません。もっと深いところにも出てきそうなのですがね、、

次回のオリパンフ号からはきちんとコンピュータを使った内容にする (予定) です。期待せずにいてください。

WSS のおはなし

竹島

PC-9821 X? や V? や C? には Windows Sound System という音源がついています[1]。なんか、DOS では使うなっていう感じの名前ですけど、面倒ですが DOS でも使えます。

[9]だからどうした！といわれるとそこまでですが

[1]Xb10 等、一部の機種を除く

まず、WSS の存在を調べます。とりあえず inportb(0xA460) & 0xF0 の値が 0x70 か 0x80 なら WSS が存在します。実は 0x60 のときもある (PC-9821Np 等) のですがこの場合は DMA #1 が使えないなど多少の制限があります[2]。

次に、気合をいれて初期化ルーチンを作ります。設定が必要な部分だけ説明します。残りの初期化はざべ 1994/12 でも見てください。(ソースコードはでかくなりそうなので省略) [3]

WSS のレジスタ 0x08 の設定

bit 7-5	PCM のフォーマット
bit 4	モノラル / ステレオ切り替え (ステレオのときは 1 にする)
bit 3-0	サンプリングレートを表す値 3 = 11025Hz, 7 = 22050Hz, 11 = 44100Hz など

bit 7-5 にフォーマットを表す値を設定します。PCM 8bit(符号なし) なら 000B, PCM 16bit(符号ありリトルエンディアン) なら 010B を設定します。(その他の設定は省略) ちなみにこの設定は Windows の WAV ファイル (リニア PCM) と同じです。

WSS のレジスタ 0x0E, 0x0F の設定

ここに DMA 転送時のサンプル数 - 1 の値を書き込みます。この値を間違えるとハマります (^^; [4]たとえば 16384 バイト転送する場合、8bit モノラルなら 16383 を、16bit ステレオなら 4095 を、というように設定します。

PC98 の DMA の設定

WSS は DMA 転送を利用しているので、ここを間違えるとかなりやばいことになります。(最悪の場合、ハードディスクの再フォーマットになる可能性もあります)

X-MATE や ValueStar では通常、DMA #1 を使います。ただ、DMA #1 は一部機種や C バスでは使用できないので、ユーザ設定可能にしておかないと危険です。あらかじめ、DMA チャンネル番号を WSS ポート 0x0F40 に設定しておきます。0x0F40 は読み出せるので、割り込み番号や DMA チャンネルはここから取得することも多分可能です。(割り込み番号は可能。DMA チャンネルは未確認。)

次は危険な PC98 の DMA の設定です。以下のルーチンは必ず割り込み禁止状態で行います。このルーチンは割り込み時にも使用します[5]ので、DPMI を使っているときは領域ロックをかけておいたほうが安全です。

[2]なんかえらそーなこと書いていますけど私もよく知りません (^^;

[3]djgppV2 用のでもいいからほしいって人は私に言うか、いぬ。BBS からダウンロードしてください。

[4]確か DMA 転送のたびにノイズが入ります。

[5]この後なるべく早くポート 0x0F46 に何か出力して割り込みフラグをクリアしておかないと、WSS の FIFO を使い切って音が止まってしまいます。

まず、DMA のレジスタを設定しているときは DMA 転送を止めておきます。ポート 0x15 に、0x04 |DMA チャンネル番号を出力すればその DMA 転送は止まります。0x00 |DMA チャンネル番号を出力すればその DMA 転送は使用可能になります。

次に、DMA 転送モードを設定します。WSSPCM 再生時の DMA 転送モードに設定するには、

outportb(0x17, 0x28 |DMA チャンネル番号) とします。

DMA のポートはチャンネル番号によって決まっていて、次のようになります。

チャンネル	バンク	アドレス	カウンタ
0	0x27	0x01	0x03
1	0x21	0x05	0x07
3	0x25	0x0D	0x0F

DMA 転送においては、バンクとアドレスで転送バッファの先頭アドレスを指定し、カウンタで指定したバイト数だけ転送します。アドレス、バイト数は 2 バイト (16 ビット) の値なので値を出力する前に 0x19 に何か出力しておき、下位、上位の順に出力します。DMA バンク、アドレスで設定する値は、24 ビットのリニアアドレスの上位 8 ビットを DMA バンクレジスタに、下位 16 ビットを DMA アドレスに設定します。これじゃよくわからないので例をあげると、1234:5678 から 16384 バイト転送するのであれば、リニアアドレスは 0x12340 + 0x5678 = 0x179B8 ですからバンクは 0x01 、アドレスは 0x79B8、カウンタは 0x4000(=16384) となります。ただし、DMA は 24 ビット (16M) までのメモリアドレスしか扱えません[6]。よって、VCPI や DPMI で転送する場合はそのアドレスが 16M 以下でなければ転送できません。これを避けるため、Windows などでは 16M 以下にバッファを用意し 16M を超えるアドレスのときは一度そのバッファに転送してから DMA 転送を行っています。(DJGPP などで使う場合は DOS メモリを確保するのが簡単でしょう) また、186 以下の 98 では DMA バンクをまたぐ転送はできません。つまり、286 のマシンに搭載されている DMA でないとまともに使えません。

これで一番危険な DMA 転送はできるはずです。残りは面倒ですがそんなにむずかしくないはずです[7]。

拡張サウンド機能 ID ポートの移動

さて、WSS ははじめに述べたとおりポート 0xA460 を ID レジスタとして使用しています。これは 86 ボードの ID とポートが重なるため、通常、同時に使用することはできません。マニュアルにもそうかいてあります。しかし、一部の機種を除くと WSS の Plug & Play 機能を使えば共存可能になります。86 ボードを使うために WSS を切り離している人は多いと思いますが、そのやり方もここに書いておきましょう。86 ボードは WSS と同時には使わないほうがいいような気もしますけど[8]。

[6] 16 M以下でも、VRAM を転送バッファに使ったりするのは危険です。
[7] 某 SRN-F などを無視すれば、の話ですけど。
[8] 使っている本人が言っても説得力ないですね (^^;

ポート 0xC24 に 0xE1 を、ポート 0xC2B(下位),0xC2D(上位) に ID ポート番号を出力すれば、 ID ポートの移動ができます。(ただし、ハードウェアの制限のため特定のポートにしか移動できません) たとえば、0xB460 に移動する場合は、outportb(0xC24,0xE1) とした後に、outportb(0xC2B,0x60),outportb(0xC2D,0xB4) とします。Windows 95 の AUTOEXEC.BAT にこれを組み込み、WSS 割り込みと 86 ボードの割り込みを重ならないようにすれば、ハードウェアウィザードで I/O ポート が重なっているという警告は出ますが、Windows 95 でも正常動作するようです。また、MS-DOS や MS-DOS モード上では 86 ボードが使えるようになります。(ポート 0xA460 から 0x4? や 0x5? が読み出せるようになります) 確か WSS を切り離している状態では内蔵 CD も使えなくなるので、多少の利用価値はあるかもしれません。

Windows Sound System for 98 使い方

なんかろくなこと書いていないので、最後にちゃんとしたのを 1 つ。PC-9821C/X/V シリーズに搭載されている WSS の仕様のうち、再生時に役に立つ部分をまとめて書いておきましょう。ただしこれが全部というわけではないです。あやしい DOS プログラマになりたい人はどうぞ:) ちなみに、これはざべ 1994/12 を元に書いてあります。

ポート	Read/Write	bit	機能
A460H	R(/W)	7-4	サウンド ID
			0110B = PC–9821Np/Nf
			0111B = X-Mate/ValueStar
			1000B = CanBe
	R/W	1-0	OPNA マスク
			00B = OPN モード
			01B = OPNA モード
			1xB = OPN(A) 使用不可
0F44H	R	7	1 = 初期化中
	R/W	6	MCE : 1 = モード変更可
	R/W	5	1 = 割り込み中の DMA 転送禁止
	R/W	4-0	CS423x レジスタ Index
0F45H	R/W	7-0	CS423x レジスタ Data
0F46H	W	7-0	任意:割り込み終了通知
	R	7-0	各種ステータス (一部割愛)
	R	5	1 = 録音データ引き取り可
	R	1	1 = 再生データ送出可
	R	0	1 = 割り込み処理中
0F47H	R/W	5-3	PCM 割り込み番号
			000B = 使用しない
			001B = INT0
			010B = INT1
			011B = INT41
			100B = INT5
	R/W	2-0	DMA チャネル
			000B = 使用しない
			001B = DMA0
			010B = DMA1(Np,Nf 不可)
			000B = DMA3
0F43H	R	5-0	WSS ID (000100B)
0C24H	R/W	7	0/1 = WSS を使用 しない/する
	R/W	6-0	PnP ポート
			1110000B = 0F40H-0F47H の再配置
			1110001B = A460H の再配置
0C2BH	R/W	7-0	再配置 I/O アドレス下位 8 bit
0C2DH	R/W	7-0	再配置 I/O アドレス上位 8 bit

ここからが CS4231 のレジスタ一覧です。

Index	bit	機能
06H	7	Left DAC MUTE(0:true/1:false)
	5-0	Left Volume 0(max) to 63(min)
07H	7	Right DAC MUTE(0:true/1:false)
	5-0	Right Volume 0(max) to 63(min)
08H	7-5	再生時フォーマット
		000B = リニア PCM 8 bit(unsigned)
		000B = リニア PCM 8 bit(unsigned)
		001B = μ-law 8bit
		010B = リニア PCM 16bit(signed) リトルエンディアン
		011B = A-Law 8bit
		101B = ADPCM(IMA) 4bit
		110B = リニア PCM 16bit(signed) ビッグエンディアン
	4	再生チャンネル数設定
		0 モノラル / 1 ステレオ
	3-0	入力クロック／分周数
		1010B = 64000Hz(保証外)
		1000B = 54840Hz(保証外)
		1100B = 48000Hz
		1011B = 44100Hz
		1001B = 37800Hz
		1101B = 33075Hz
		0110B = 32000Hz
		0100B = 27420Hz
		0111B = 22050Hz
		0101B = 18900Hz
		0010B = 16000Hz
		0011B = 11025Hz
		1110B = 9600Hz
		0000B = 8000Hz
		1111B = 6620Hz
		0001B = 5510Hz

09H	7	PCM 再生 (0:停止/1:有効)
	6	PCM 録音 (0:停止/1:有効)
	3	1 : MCE を 1 から 0 にする時、自動調整を行う
	2	0/1 (single/dual(PC98 不可)) DMA チャネルモード
	1	再生転送モード (MCE = 1 の時のみ設定可能)
		0/1 (DMA/CPU) 転送モード
	0	録音転送モード (MCE = 1 の時のみ設定可能)
		0/1 (DMA/CPU) 転送モード
0AH	7,6	TTL ロジック 1,0
		XCTL1,0 端子のロジックレベル設定
	3	8bit 録音ディザリング許可フラグ
		0/1 ディザリング (禁止/許可)
	1	0/1 割り込み (禁止/許可)
0CH	7	常に 1
	6	MODE2 フラグ, 0/1 (CS4248 互換/CS4231 拡張) モード
	3-0	Revision フラグ
		0001B = CS4231 Revision ”B”
		1010B = CS4231 Revision ”C” 以降
0EH	7-0	DMA 転送 PCM データ長 (下位)
0FH	7-0	DMA 転送 PCM データ長 (上位)
		サンプル数 – 1 を設定する
10H	7	アナログ出力レベル
		0:2Vpp(-3db) /1:2.8Vpp(0db)
	6	タイマ割り込み使用許可フラグ 0/1 使用しない／する
	0	D/A コンバータ設定 0/1 有効/0 レベル出力
11H	0	ハイパスフィルタ 0:有効／ 1:無効
1CH	7-5	録音時フォーマット (08H と同じ)
	4	録音チャンネル数設定 (同上)

編集後記

いやはや、見てわかるかどうかわかりませんが、今回の部報から LaTeX2.09 から pLaTeX 2_ε に変えました。それに伴い、先代編集長のマクロを参考にしながらとはいえ全てをはじめから書き直しました。
まだまだ、完成には程遠いながら締め切りが来てしまったので今回はこのまま印刷に回しますが、次回以降はさらに煮詰めていくと共に多少見た目も変えていこうかと画策しています。
あぁ、表紙は部報マクロの作成状況を的確に表したものです。某アニメとは何の関係もありません。念のため。
さて、次回部報はオリパンフ号です。原稿提出手段は近日中に連絡します。各分科会責任者は原稿を用意して待っていてください。

理論科学グループ　部報 205 号

1997 年 3 月 8 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 205 号
— 追い出しコンパ号 —
1997 年 3 月 8 日

THEORETICAL SCIENCE GROUP