WORD

2009.10 From College of Information Science

WORD 政権交代号

11

特別インタビュー
ピクシブ株式会社取締役
永田寛哲氏

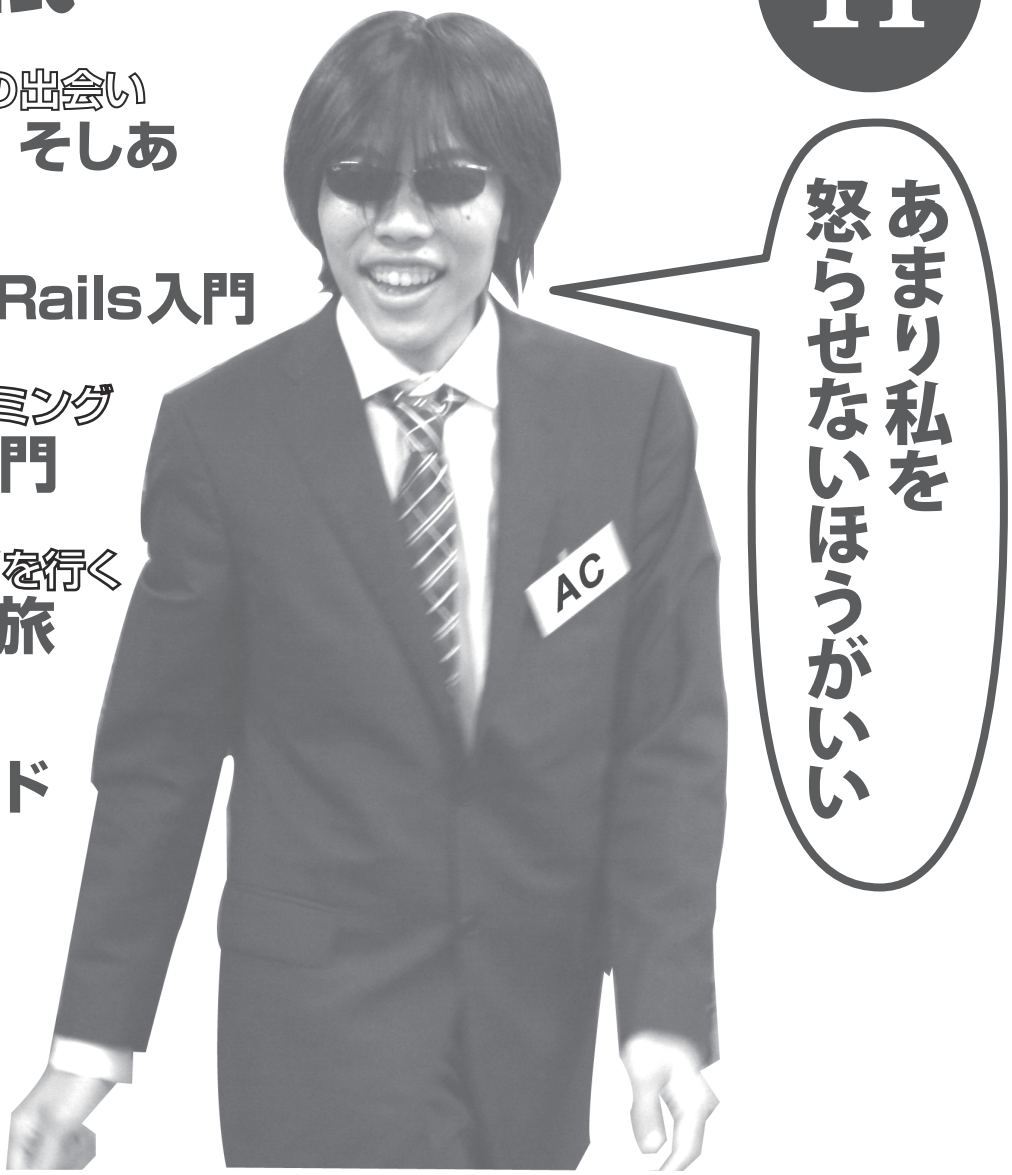
社会学類誌との出会い
WORD × そしあ

連続掲載
Ruby on Rails入門

GPUプログラミング
CUDA入門

サービスエリアを行く
SAぶらり旅

神ゲー?攻略
バトルトード



Vol. 11

2009年10月号

CONTENTS WORD

pixiv インタビュー

文 編集部　ろんず、少佐、Genyakun、Santa-、Flast、なるがみ

はじめに

近年、日本ではアニメやゲームなどのいわゆる「ヲタク文化」が注目されています。ニュースやバラエティ番組で特集されたり、ヲタク文化を題材とする映画が大ヒットするなど、話題に事欠きません。

今回は二次創作[*1]という活動に着目しました。現在、インターネットの発達によって二次創作のスタイルが変わりつつあります。昔は友人知人同士に限定されていた活動が、今ではインターネットを介して世界中の人と行うことが可能になりました。

それにより今まででは考えられなかったような多種多様なジャンルや流れが生まれました。個人が作ったゲームキャラクターが、多くの人の目に触れることなど今までなかったことです。更にさらに、個人の制作作品が人気を博し、二次創作も多く生み出されています。

インターネットによって拡大した「二次創作」において、現在最も注目されているウェブサービスの 1 つが「pixiv」です。「pixiv」は絵に特化した SNS で、描く人と見る人、両者が使いやすいサービスが提供されています。これにより、今まで描き手のウェブサイト上に掲載されていた絵が集約され、様々な絵をクリック 1 つで簡単に探すことが可能になりました。見る人にとっては簡単に好みの絵を探すことができ、また、描く人にとっては絵描きさん同士の情報交換がしやすくなるなどの利点を持っています。

今後、インターネットを活かした二次創作文化に根ざしたウェブサービスが更にさらに登場することでしょう。いずれ、オンライン上で仮想同人誌即売会が開かれることもあるかもしれません。

pixiv インタビュー

編集部は、「pixiv」を運営しているピクシブ株式会社の取締役副社長である永田寛哲さんにインタビューを行いました。インタビューの中で新たに分かった「pixiv」が目指す姿をお伝えします。

*1　詳しくは Wikipedia の「二次創作物」を参照
http://ja.wikipedia.org/wiki/%E4%BA%8C%E6%AC%A1%E5%89%B5%E4%BD%9C%E7%89%A9

■　「pixiv」は偶然の繋がりから始まった

編集部（以下、編）

なぜ絵をテーマに「pixiv」を始めたのですか？

永田寛哲さん

永田寛哲さん（以下、永田）

プログラマの馬骨さんという人が趣味でサービスを起ち上げたんですよ。入社以前にも彼にはよく仕事を頼んでいました。

当時は TINAMI[*2] のようなウェブサイトを代表として、リンク集などから絵描きさんのサイトに行く必要がありました。絵描きさんのサイトの更新をまとめた個人サイトがあったくらいです。もっと効率よくまとめれないか考えたのが始まりでした。

最初は馬骨さん名義のリリースでしたが、すぐに盛り上がっちゃいましてね。個人サーバで「pixiv」を運営することが難しいため、馬骨さんに入社してもらい、会社の事業として「pixiv」を運営することになりました。

編

今のプログラマの方が絵が好きで始めたのがきっかけだったのですね。

永田

この出会いがきっかけで一緒に仕事をするようになりました。馬骨さんと出会え、それをきっかけに「pixiv」が誕生したのは運命だと思っています。

*2　http://www.tinami.com/

■　プログラマ＋絵描きさんだから実現できた

編

なぜ、絵描きさんと見てる方とのコミュニケーションに着目したのですか？

「pixiv」が絵を使ったコミュニケーションに特化していることは、「pixiv」を触るとすぐに分かります。絵を集約するだけでなく、絵描きさんも一緒にサービスの中に取り込もうという姿勢が表れていると思います。

永田

・　馬骨さんの理念

馬骨さんの基本理念に、「どうすれば絵を描く人が集まるか」ということがあったと思います。ウェブサービスは器だけ作ってもユーザが集まらなければ意味がありません。逆にいえば、集まることさえできれば良いサービスであり、集まらなければどんなに良い物でも酷いサービスになってしまう側面があると思います。その点では、描く・見るの両方の心を馬骨さんが分かっていたことが大きかったですね。

・　一目見て

完成した「pixiv」を見た時、ユーザの心情をとらえていると感じました。

僕自身もかなりヲタクで、コミケに 15 年くらい行ったり、元々その分野の仕事をしてたんですよ（笑）。だから絵描きさんが何を欲しているか理解しているつもりです。彼らは目新しいとか少し新しい物に食いつく、いわゆるギークとは違いますよね。だからサービスとしてどんなに良くても、彼らが盛り上がってくれないと話にならないと思っています。

・文字を排除した絵中心のコミュニケーション

永田

凄いなと思ったのが、日記もなく、コメント欄もデフォルトで非表示にするなど、思い切ってテキストを排除して絵を際立たせていることです。絵描きさんが「ここは絵で純粋に勝負できる場だ」と思えるような設計というのですかね。

編

積極的にテキストが排除され、絵が主体のサイトであると見てて凄く伝わってきます。

永田

そう！　それが絵描きさん的にも受けが良いみたいです。「あっ、ここは絵で勝負する場所なんだな」っていうのが分かりますからね。

文字のコミュニケーションは、自分のサイトの掲示板や、mixi やメッセンジャーでも可能です。その中で、「pixiv」は「ここは絵を見る場所なんですよ」「絵をアップする場所なんですよ」っていうポジションなんです。

絵中心のコミュニケーションには効用があって、文字がないから炎上やケンカが発生しにくいんです。人間の怒りって瞬間的なものですから、文字に起こすと繰り返されますが、絵を使って怒りを表現するのは描いてて面倒臭いですからね（笑）。

また、絵描きさんは絵を説明するのに mixi やブログなんかを使っています。そうすると、「pixiv」だけで世界が閉じられることがありません。狭いコミュニティの仲間に見てもらえば満足している人も、「『pixiv』にアップロードしたよ」って皆に知らせることで「pixiv」が促進されるんです。

■　サービスを始めて変わった絵描きへの意識

編

サービスを始めてから感じた、絵と絵描きさんに対する意識の変化はありますか？

永田

「pixiv」が良いサービスだとは思いましたが、こんなに人が集まるとは思わなかったので驚いています。さっきの通り、僕自身は保守的なため、最初「pixiv」は駄目だと思っていました。絵描きさんは、よほどのことがない限り、自分の活動範囲を広げないと思っていたんです（笑）。

でもやっぱり保守的な人もいます。有料化サービスを始めた時、投げ銭[*3] という言葉に対して拒否反応を示す人もいました。お金のやり取りを嫌がる人もいますが、「コミケ」なんかは何度もお金をやり取りしますよね。それは別物だっていうチャンネルが入っているんだと思います。3 ヶ月もすると騒がなくなりましたが、要は環境が変わることが嫌なんじゃないのかと思っています。

■　pixiv のテーマ～「お絵かき楽しす」～

永田

よく使っている言葉が「お絵かき楽しす」です。絵に対する敷居をどんどん下げていきたいとサービスを始めてから徐々に固まってきた理念です。

絵を描くことは楽しいですよね。絵を発表していく場として「pixiv」の規模も影響力も大きくなれば、初めて絵を描いた場所が「pixiv」だったという人や、「pixiv」が絵を描き始めるきっかけになったという人が出てくると思います。そういう人が増えていくっていうことが凄く嬉しいです。

1 ヲタクというか 1 漫画好きとして、絵を描き始める人がぜひ生まれてきて欲しいし、それができれば本望だと思ってます。

*3　ポイントを他のユーザへプレゼントすることができ、そのポイントを換金できる

編

「pixiv」が絵を描ける人のコミュニケーションの場から、絵描きさんを創出する場に変わったという印象を受けます。

永田

そうですね。歌におけるカラオケボックスみたいに、「pixiv」が絵におけるそういう場になっているのだと思います。

だから今は敷居を下げることを特に考えています。業界としての使命といえば大げさですが、これだけ影響力を与えることのできるサービスになった以上、将来的にそこまで手を広げられたらと考えています。

編

現在「pixiv」の絵描きさんと見る人の割合はどうなっていますか？

永田

1 枚でも絵をアップロードしたことのあるユーザが全体の約 3 割です。実際にはアクティブに絵を描いているユーザはもっと少ないですが、1 枚でもアップロードしたユーザはたいていアクティブですね。

絵をアップロードしないユーザ全てが絵を描かないわけじゃなくて、様子見で参加してる人や、プロだから自重してる人もいると思います。

編

今まで絵を描いてなかった人が突然描き始めることはありますか？

永田

初めての絵を「pixiv」にアップロードするユーザはいますよ。僕がリアルに知ってるだけでも何人かいます。ユーザから声を聞くことも多いです。

■ 「pixiv」から始まる新たな可能性

編

ユーザ数が増えると、様々な思考を持つ絵描きさんが集まると思います。
今まで実現できなかったこととして、例えばマイナージャンルの発掘があると思います。

永田

僕もそれはビックリしました。マイナージャンルは業界での需要が早い段階からありました。でも出版社の人が、そういう絵を発注したくても昔は難しかったんですよ。それが「pixiv」ができてから飛躍的に楽になりました。

編

今でもマイナージャンルに関して、「pixiv」を使ってオファーする方はいらっしゃいますか？

永田

マイナーな所だけでなく、メジャーな所もオファーが進んでいます。仕事の関係で業界の人との交友もありますが、今は「pixiv」で探しているようですね。『新人は「pixiv」で探す』が合い言葉です。

数年前にはコミケが担っていた新人発掘の場が、今では「pixiv」になりつつあります。それによって、「頑張れば pixiv でデビューできる」などのモチベーションにも繋がります。そうなれば、「絵を描こう」「上手くなろう」という人が増え、「pixiv」にもっとユーザと絵が集まるという好循環が生まれて欲しいと思います。

■　活動と外に繋ぐ「pixiv ブログ」

編

「pixiv ブログ」というブログサービスを開始されてましたね。「pixiv」自体とは混ぜない、あくまでサブの位置付けで始めたのですか？

永田

僕が提案して最初は反対されましたが、僕は「pixiv」の出口として「絵を見せる」ことに特化したブログサービスを用意してあげたかったんです。絵描きさんにはネットに繋ぐのすら一苦労みたいな人が結構います。その中で、ブログで簡易的にウェブサイトを用意する人が最近かなり増えていましてね。

便宜上ブログとして使っていますが、僕は「pixiv」の外に向けて絵を発表できる場を作りたかったんですよ。

編

ブログと聞いた時、文字ベースかなと思いました。

永田

見てもらえば分かりますが、絵を見せることに注力したデザインになっています。

僕もビックリしていますが、「pixiv ブログ」は結構使われています。今[*4]3 万ユーザくらいで、1 日 1000 エントリー以上の規模になっています。開始一ヶ月でこのレベルまで行くとは僕自身思ってもみなかったことです。潜在的に需要があったのだと思いますね。

*4　2009 年 7 月　時点

■　文化発信の担い手として、「pixiv」の世界を見据えた取り組み

編

海外のウェブサイトで「pixiv」が紹介されているのを見かけることがあります。海外からのアクセスは増えているのでしょうか？

永田

増えてます。だいたい 5 %という表現をしていますが、ちょっと前までは 4 %で、今は全体の 7 %に近い状態です。

編

アクセスの増加を意識していますか？

永田

意識していますよ。僕らは最終的に「世界で」ということを考えています。僕と社長がこの会社を始めた時の目標が、「世界で成功するウェブサービスを作ること」でした。ましてや、今の「pixiv」が「世界で通用するコンテンツを扱えている」ことは最大のチャンスです。それはもう、強く意識していますよ。

世界に向けてローカライズすることは簡単です。ただ、まずは日本で基盤をおさえてから海外へ展開するつもりです。障害や問題が残ったまま世界へ出ても効率が悪いですし、コンテンツフォルダーや他媒体との関係性などの問題をクリアすることが必要です。そのため、現在はあえて足元を固めています。

■　同人と「pixiv」、互いにもたらした影響

編

現在「pixiv」にはタグ数 30 万の東方や、6 万の初音ミクがあります。なぜ「pixiv」で東方や初音ミクが流行ったと思いますか？

永田

まず初音ミクが発売されたのが 2007 年の 8 月末です。「pixiv」が始まったのが同年 9 月 10 日です。初音ミクって発売してすぐにガガッときましたよね。そのブームに乗ることができたというのがあると思います。

更に、グーグルで初音ミクの絵が表示されないって事件[*5] があったのを覚えていますか？でも「pixiv」では普通に見られたので、ミク難民が「pixiv」に来たんですよ。だから、初音ミク

*5　2007 年 10 月 15 日頃、クリプトンが削除依頼を出していないにも関わらず、Google イメージ検索で「初音ミク」の結果がヒットしない現象が発生した。

やボーカロイドに強いのは、その関係でしょうか。

東方ってのもとても面白くて、二次創作から三次、四次と勝手に分裂していきますよね。その状況って「pixiv」でいう「イメージレスポンス」と近い概念で、創作が創作を生んでいるのだと思います。そういうのが東方の面白さであり発展してきた方法なので、それが「pixiv」の設計に合っていたのかもしれません。

■　「pixiv」で生まれた「pixiv 文化」

編

絵を中心としたコミュニケーションが発展する発信源として、「pixiv」から生まれてくる流れについてどう思いますか？

永田

我々の運営理念として、文化を創りたいと思っています。

さっきいったように、「pixiv」がなかったら描かれなかった絵は多く存在していますし、「pixiv」が絵を描くモチベーションになっています。

昔はこんなにネットに絵がなかったと思います。描ける人でも、「即売会のような締め切りがある時だけ集中して描く」というイメージから、「pixiv」があるから、ランキングにチャレンジしたり、毎日見たり、感想を言い合う、24 時間営業な場になってます。

投稿される絵もたくさん増えましたし、大げさだけど文化を創っているのかもしれません。

「pixiv」がきっかけで絵を描き始め、そして有名な漫画家になれば、その流れは文化になると思います。だから、絵に対する表現について、絵を発表したい、描きたいと思わせる方向へ奨励して行きたいと思っています。

■　「pixiv 文化」をリアルに出すということ

編

「pixiv」で生まれた文化のアウトプットとして「pixiv　FESTA」を開催されていますよね。足を運んで見るというスタイルを選んだ理由はありますか？

永田

単純に面白そうだったこともありますが、僕の持論として「ウェブとネットの人格を切り分けない」ことが 1 番の理由です。私が昔の仕事でデビューしたきっかけもウェブの人脈によるものでした。友達もたくさんできましたし、リアルが豊かになった実感があったんです。

それこそがウェブ・インターネットの醍醐味であり、核心的な所だと思っています。だから現実とインターネットを切り分ける必要はありません。もっといえば「ネットはリアルに従属

されるべき存在」という考え方です。

ネット大好きですが、それはリアルを豊かにするためだと思っています。ネットで盛り上がったら、どんどんリアルに行って欲しいですね。

編

なるほど。確かに私もそういうのが結構好きで、気に入った絵は印刷したりしますね。

永田

そうそう。皆作るの好きだから、現物になると嬉しくなるし、それをみんなで楽しみます。

「pixiv　FESTA」でもマイピク同士が初めて会う光景を見られてよかったです。こういう中から、例えば結婚する人が出てくるかもしれません。そんな場を提供できるのが嬉しいです。

編

なるほど、分かりました。
長くなりましたが、ありがとうございました。

終わりに

今回、インタビューで「文化を創る」「世界を目指す」という 2 点に注力されていることが伝わってきました。

文化を創るという点では、絵を描く人たちの心を掴んだ「積極的なテキスト排除」や、様々な絵を描く人たちを集約したことによる「新人発掘の場」といった点が「pixiv」の魅力ではないでしょうか。使う人の立場を考えないと、人が集まるコミュニティを形成できないことを教えて頂きました。

ぜひ「pixiv」には世界を開拓してもらって、「絵を描く事は楽しい」という輪を広げていってもらいたいです。

日本の経済をも左右させるといわれるこのヲタク産業を支える「pixiv」にインタビューを行った記事を書いてみました。皆さんはどのような感想を持たれましたか。この記事をきっかけに、二次創作もしくは絵の世界に足を踏み入れる方がいれば記事を書いた甲斐があったものです。

順序立てて分かりやすく説明してくださったピクシブ株式会社取締役副社長の永田さんに、この場を借りて、厚くお礼を申し上げさせて頂きます。ありがとうございました。

WORD × そしあ

文 編集部（いのひろ、一七夜月）

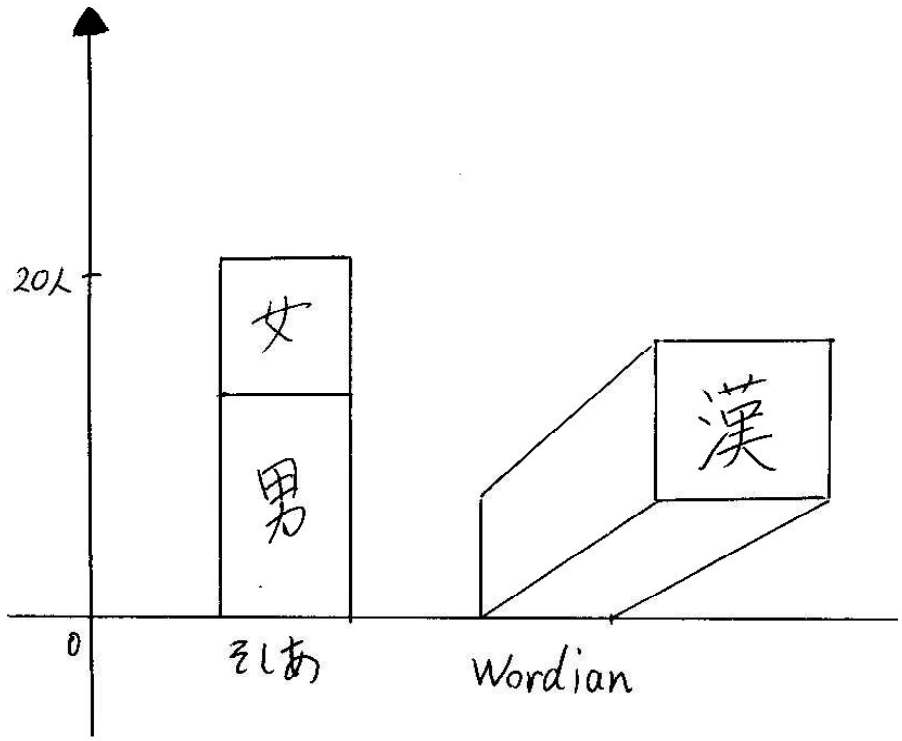
「そしあ」とかいう連中がいるらしい。筑波大学の絶滅危惧種「学類誌」。その大御所「そおしあ〜る」。なにげに気になるそのメンバーと対談する機会を得た。会いに行ってやろうじゃないか。

そおしあ〜るとは

社会学類誌そおしあ〜る（そしあ）とは、社会学類誌委員会によって制作されている数少ない学類誌の一つだ。社会学類誌ということもあってとなりのトトロの穿った分析から怪しいカレーの作り方まで何でもござれ。制作された雑誌は 1C201 室の前で配布されているぞ。

そしあが創刊されたのは、1981 年 6 月 1 日。当時はまだ「そしあ」の名はなく、「社会学類誌」という~~先鋒な~~名前だった。ちなみに我が WORD の創刊は 1979 年 1 月 26 日。偉大なる高橋女史により WORD は産声を上げた。筑波大学ができたのが 1973 年だから WORD の歴史は筑波大学の歴史といっても過言ではない!

現在 WORD 部員は 25 人。男女比何と 25:0。対してそしあは男女比 13:8 の 21 人。~~うらやましい~~でも WORD だって女性部員はいっぱい居たんだからね!!11!



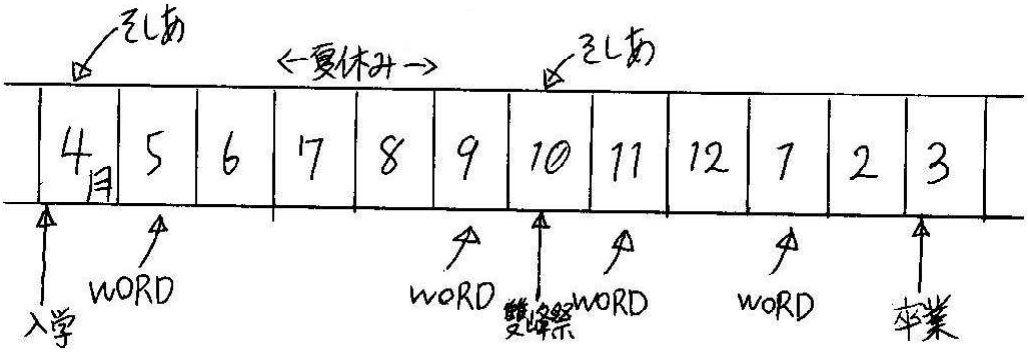
グラフで比較すればそれほど差はない。むしろそしあ民の方が多く感じられる。

・WORD ができるまで

各員が書きたい記事を思いつく。その後、編集者会議で所信表明。期限までに第一稿を完成させ、編集者全員で赤入れ。一週間後の期限までに完成版を仕上げる。完成版を集め、ヘッダ、フッタを統一する。表紙、裏表紙、目次、編集後記などを含め PDF 化し、マスターと呼ばれる原盤が印刷される。マスターは通常のプリンタで印刷されるが、量産は学類のリソグラフを使う。高速なリソグラフだが、量産が終わるまでに半日から一日程度かかる。量産が終了した暁には、編集部員の手によってソートされ、冊子の形に整えられ、ホチキスが穿たれる。翌日、もしくは夜の間にひっそりと実戦配備され、読者の手に渡ることになる。

・そしあができるまで

草稿集という極秘資料が存在し、各員がネタを書いていく。ネタを書いた人は、ネタに Go サインが出たらレジュメを提出し、評価する。これによりページ数が割り振られる。記事の提出期限には”半省会”と呼ばれる会議が催され、赤入れが行われる。完成した記事は、FAX 用紙に写真とともに貼り付けられ、B4 用紙に印刷される。印刷された版は編集部員による人海戦術によって半分に折りたたまれ、冊子の形に仕上げられる。この「折り」には、そしあ民の血と涙によって受け継がれている伝統であり、誇りである。



年間発行スケジュール

WORD × そしあ

そおしあ～るとの対談

-お互いの記事について-

WORD 編集部員(以下 W)：そしあさんから見て、ぶっちゃけ WORD をどう思いますか？

そ～しあ～る編集部員(以下そ)：専門性が強くて、コンピュータ専門誌のように見えますね。ただ、時々 MAX コーヒーとか、バトルトードとかネタ記事もあって、そういった部分はそしあにもにていると思います。でも、謎な記号で埋め尽くされた記事が結構あって、何が書いてあるのかよく分からないことも多いです……。情報科学類誌ということもあって、情報の人たちは普段からこんなことをしているのかと思うと恐ろしいです。

W：実際にはそんなことはないんですよ。WORD の記事にあるようなことは授業でやることもほとんど無くて、実際情報の人間でもどれだけの人が理解しているのか(笑)

そ：そうだったんですか(笑)　共通科目の情報の実習では、今後使うことの無いようなアルゴリズムだとかをやらされてアレルギー気味なので、WORD の記事に書いてあるようなことは試そうとは思わないですね。

W：逆に、私たちがそしあを見ても全く読んで分からないような記事はあまりないですね。そういった意味ではそしあは優れているのではないでしょうか。ネタ記事に関してはコンセプトというか、やっていることは違っていても、その根幹は同じというか、なんていうのかな。

そ：バカを真面目にやる？

W：そう、まさにそうです。どうでもいいようなことを真面目にやるっていうのはネタの基本ですよね(笑)

そ：でも、純度の高いネタを保っていくことは大変です。

W：わかります。そしあさんは毎年恒例の記事とかがありますよね。

そ：センタープレですか。あれも毎年苦労しているんですよ。伝統といえば聞こえがいいですが、マンネリ化しないように必死なんですよ。新入生にだけ受けていてもしかたないので。WORD にはそういった恒例の記事はないのですか。

W：無いですね。引っ越し準備号とか、入学祝い号には毎年同じような記事が載りますが、その他では毎年やっていこうみたいな企画は無いですね。MAX コーヒーネタはそういった地位を占めつつありますが、今後どうなることやら(笑)

そ：期待していますw。ほかにも、WORD の統一された表紙なんかは良いですよね。サブタイトルとか時事ネタが絡んできて非常に面白いです。

W：ありがとうございます。そしあさんは一冊一冊のまとまりがあって良いなぁと思ってます。WORD だとそれぞれの記事はそれぞれの筆者が思い思いに書くためまとまり感はどうしても出にくくなってしまって……。

そ：制作過程の違いが出てますね。そしあは作るときにページ割り[*1] を先に決めるなどレイアウトなんかを重視してますから。

W：WORD とは真逆ですね。WORD はそれぞれの記事ができあがってからページ数を調整して、中途半端にページが余ったら書籍紹介を突っ込んだりして(笑)

そ：なるほど。あと、レイアウトがらみだと WORD はオチを付けるときに改ページするみたいな大胆な構成を良くされますよね。こっちではあまりページに白いところを残してはいけないみたいな風潮があってなかなかそういった大胆なことはしづらいです。

W：WORD にはそういった考えはあまりないですね。プログラムとかをよく書くので空白や改行をあまり気にせず使えるからでしょうか。

-何故編集部に入ろうと思ったのか-

W：合格が決まると引っ越し準備号が送られてくるのですが、そこで WORD の存在を知る人が多いみたいです。ただ、WORD 編集部員の中にはそれ以前から知っていたという人も若干いますが(笑)　今年度の新入部員も例に漏れず新歓以前から WORD にいた人が多いような多いような。でも今年の新歓は空前の集まり具合でしたね。

そ：我々は逆にあとから人づての方が多いですね。松美記念[*2] に参加したら、いつの間にか編集スタッフになっていたりとか。あと、今の編集部員にはそしあ二世がいます。親子そろってそしあです(笑)

W：それは凄い。WORD ではそういった話は聞いたこと無いです orz

そ：まぁ、いろいろありますがそしあに入って良かったと思ってます。そしあに入ると留年するみたいな都市伝説があるみたいですが、縦のつながりもけっこうできたし、記事を読んで喜んでもらえたり、話しかけてもらえたりするのはとてもうれしいです。

W：それは私たちも同じです。WORD には奇特な人が集まると一部では揶揄されているようですが個性あふれる人がいっぱい居て、人脈もできますし、やっぱり記事を読んで喜んでもらえるのは何事にも代え難いですよね。

---

*1　ページ割り：どの記事にどれだけのページ数を割くかを決めること。冊子全体のレイアウトを決める重要な作業。

*2 松美記念：人が馬の仮装をして行う競馬大会。雙峰祭中に開催される。

そ：たいへんなこともいっぱいありますけどね。そしあには伝統的に「折り」という作業があるのですが、これが非常に大変で、でも誇りを持ってやっています。

W：本当に凄いと思います。折らない WORD 流の綴じ込み作業でも苦労の連続なので、そしあの「折り」には本当に脱帽します。

-学類誌について-

W：昔はそしあや WORD 以外にも学類誌が有ったようですが、今は少なくなってしまいました。そのことに関してはどう思われますか？

そ：むしろ他の学類誌が無くなってしまった理由が気になりますね。やはり人がいなくなってしまったのでしょうか。そしあに関しては理解がある先生方がいらっしゃって残させていただいている面もありますね。

W：WORD もそういった面はあると思います。

そ：メディアに関心があったり、発信したい人が多いのかも。

W：ブログでやれというような意見があるようですが、それに関しては。

そ：アンケートによく有りますね。しかし、やはり紙媒体では重みが違うと思います。責任を持って出せますし。

W：そうですね。同じような理由で我々も紙媒体にこだわってます。一種の誇りですねｗ。

そ：そしあは自慰行為にたとえられることが有ります。変な表現になりますが、読者と愛のある（性）交渉をしたい。

W：その辺りは積極的ですね。アンケート作ったりとか。WORD も愛のある交渉をするために見習っていきたいですね。

そ：アンケートの回収率から行けば、険しい道ですけどね。努力していきたいです。

W：お互いに協力し合えると良いですね。共同で企画やネタをやってみたりとか。

そ：望むところです。ところで、もうじきセンタープレという恒例イベントが有るのですが……。

W：WORD 参戦ですか。受けて立ちますよ(笑)

サービスエリアぶらり旅（関東～関西）

文 編集部 ミレトス

はじめに

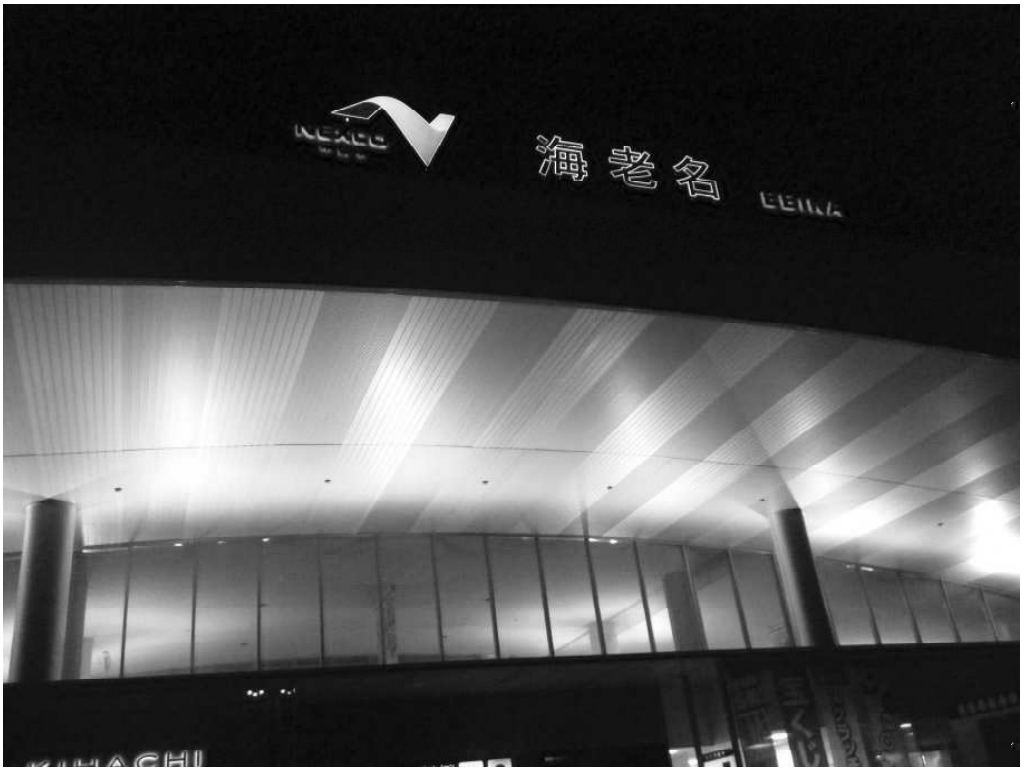
皆さん初めまして。ミレトスといいます。とうとう夏休みが終わりましたが、有意義に過ごす事はできたでしょうか。**私は浪費してしまいました。**まあそうはいっても、ずっと引きこもっていたわけではないのです。今年は久しぶりに祖父母の家へ遊びに行きました。家族全員で行くので、高速も安くなったということもあり、高速道路を使って行きました。走行距離は実に**600km 超え。**埼玉から兵庫への長旅です。今回はそんな長旅の休息地、サービスエリア（以下 SA）の魅力を紹介していきます。

東名高速道路編

埼玉から兵庫まで行くには、まず首都高速を抜けなければなりません。ここは渋滞の名産地でいつも渋滞ばかりなのですが、今回はすんなり通れました。そして、首都高速を抜けると東名高速に入ります。すると 1 番目の SA が見えてきます。

●海老名 SA

記念すべき 1 個目の SA は、神奈川県にある海老名 SA です。



SA ぶらり旅

SA には長旅に必要な物が色々と揃っています。食べ物からお土産、ガソリンスタンド等々、旅にはかかせないものばかりです。ここ海老名 SA にはうまいもの横町というものがあり、様々な食べ物を売っています。神奈川なのに大阪名物とかね……。

中はこのようになっており、大抵 24 時間営業しています。海老名 SA の売りはメロンパン。テレビで紹介されるほどの代物らしいです。またここでは東京のお土産を多く扱っており、関西地方へのお土産を買うにはうってつけです。

さて、高速道路を多く利用する方なら分かると思いますが、運転手が SA に寄るのは休憩だけが目的ではありません。実は SA には高速道路の交通情報があり、これがこの先走っていく上で非常に重要なのです。どこで事故があって何キロの渋滞があり、どこが工事中で通行止めなのか、などといった情報が画面に映し出されています。さすがに深夜２時にもなると人は少なかったですが……。

●牧之原 SA

次に到着した場所は静岡県にある牧之原 SA。やはり静岡ということもあり、お茶や抹茶味などの商品が多く見られます。そしてもう１つ、静岡と言えばやはり富士山を思い出す方が多いと思います。東名高速ではこの牧之原 SA の１つ手前にある富士見 SA から眺める事が出来ます。明け方や夕方などに行くと中々いい景色が見られますよ。

この SA で興味をひかれたのは、緑茶セットというメニューです。ただ緑茶を飲むだけではなく、緑茶の入れ方まで教えてくれるそうです。値段も手頃なので時間があれば注文してみたいですね。

牧之原って一応読める･･･よね？
下の画像は白黒なのでわからないかもしれませんが、やはり緑色の包みが多いです。抹茶プリンやお茶っ葉の詰め放題などまさに緑一色！

牧之原 SA を出た後、東名高速で一番大きい浜名湖 SA に行く予定だったのですが、私の SA 観光のしすぎで到着時間がかなり遅れそうということで寄りませんでした。**この時点では帰りに寄る予定だったんです。**では何故ここで書かないのか……？　それはこの SA ぶらり旅を最後まで読み進めればいずれわかることでしょう。

名神高速道路編

東名高速も終わり、名神高速道路へ。しかし目的地が近かったので 1 カ所しか寄れませんでした。

●吹田 SA

この兵庫への長い旅ももうすぐ終わり……最後に寄ったのは大阪府にある吹田 SA でした。

これを撮影したのは午前 9 時頃。気づいたら明るくなっていました。大阪といったらたこ焼き。たこ焼きといったら大阪でしょう。もちろん売っていますとも。あとここの SA は近畿地方周辺のお土産も色々置いており、滋賀県や京都のお土産などもありました。

また、吹田 SA は上りと下りの SA を行き来できるように橋がついています。このように行き来できる SA はあまりないので少し驚きました。しかし使うのだろうか……。

あとがき

SA を巡る旅、いかがだったでしょうか。結局、自宅を深夜 0 時に出て、祖父母の家に着いたのはほぼ正午でした。今回はほぼ 12 時間かかりましたが、前は 9 時間ぐらいで着いたような……？とりあえず家族のみんな**すみませんでした orz**

今回は埼玉から兵庫へ、というルートで行きましたが、実は帰り道、**兵庫から埼玉への SA 巡りもございます。**「内容ほとんど変わらねぇじゃねえかよ！」と思っているそこのあなた。もう一度今年の夏起こった事を思い出して欲しい。埼玉を出たのが 8 月 6 日、帰ってきたのが 8 月 13 日ですよ。……そう。

東名高速道路通行止め

ということで、図らずとも違うルートを通って帰ってくる事になりました。これが理由で浜名湖に寄れなかったのが残念ですが…。結局、帰り道は中央高速道路を使いました。次回があれば帰宅編も書こうと思います。

たたかうかえる

闘蛙

書いた人： 編集部 ふぁい

■あいさつ

連載と連載の間が大きく開いてしまい、**富樫状態**と化しているこの連載も、ついに 3 回目を迎えました。1 面、2 面の攻略に続き、今回は 3 面の攻略をします。

■ Stage3:俺のピン争奪戦

この面はボーナスステージです。強制横スクロール面なので、瞬発力が要求されます。

何かよく分かんないカーリングのストーンのようなもの(下の画像参考)に乗ったバトルトードが、**何かよく分かんない**けど床に配置されているピンを次々と回収して行くという**何かよく分かんない**面です。ピンの回収率に応じて残機が増えます。

が、やすやすとボーナスを与えるほどレア社は優しくありません。ピンの回収率が良くないと残機は全く増えず、**ボーナスっぽいお恵みは何も受けられません。**

ただし、この面で残機が増えない事はあっても**減ることはありません。**このゲームにおいては、**死なない事がもはやボーナスと言っていいのかも**しれません。

ボーナス面でも鬼畜精神を忘れないレア社に乾杯！！１１！

話をレア社から 3 面に戻しましょう。この面は、当時(1993 年)にしてはかなり綺麗で繊細な疑似 3D っぽい描写が成されています。なんでこの面だけこんな高度な技術が使われているかはナゾです。**何かよく分かりません。**

そんな~~無駄な~~高度な技術のおかげで、これまでのように MAP が作成できないので、今回は MAP はありません[*1]。でも大丈夫！ **死ぬことはないから！！**

■ゲーム画面

右の画像に写っているボーリングのピンのようなものを回収していきます。回収した本数は画面左上(バトルトードのステータスの下)に表示されます。

ピンの配置や床の模様を見れば分かるように、奥行きのある画面が表現されています。手前のものは、奥のものに比べて早く動きます。いわゆる遠近法です。

さりげなく床にバトルトードやピンが反射しています。芸が細かいですね。

背景がチェスの駒と宇宙という不思議な組み合わせです。**何かよく分かりません。**

*1 面倒くさいわけじゃないよ。

闘蛙

■敵キャラ(など)の紹介

◆ピン(1 ポイント)

これを取ると、1 ポイントのボーナスポイントが得られます。200 ポイントで 1UP します。

2 人プレイの時は、我先にとこれを奪い合う**争奪戦**が展開されます。

◆ピン(5 ポイント)

なんと、これを取ると 5 ポイントのボーナスポイントが得られます。

設置数が少ないので、2 人プレイの時は**熾烈な争奪戦**が展開されます。

◆ダメージピン

これを取ると HP が 1 ポイント減ります。

HP が無くなるとボーナスステージが終了してしまいますが、残機は減りません。

◆何かよく分からない敵

こいつに当たると、ボーナスポイントが 5 ポイント減ってしまいます。

ダメージは受けません。

どうやっても倒せません。

■操作説明

上下左右キーで移動ができるほか、B ボタンでパンチが繰り出せます。

ただし、**ピンや敵には、パンチは一切当たりません。**

2Players　A モード(仲間同士の攻撃が当たるモード)でプレイしたときだけ、仲間を殴る事ができます。

ボーナス面に友情破壊機能を実装するレア社に乾杯！！１１！

■この面の見所

何度も書いているように、この面は残機が減ることはありません。

また、ピンの回収具合によっては残機が増えると書きましたが、どんなに頑張っても 500 ポイント程度しか集まらないので、2 機しか増えません。

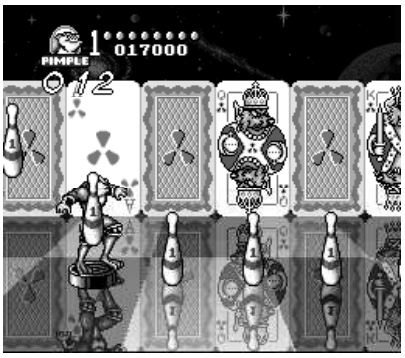
前回攻略した 2 面で、蜂の死骸を殴り続けて残機を増やした方が効率的です。

つまり、この面はきわめてどうでも良い！

というわけで、今までのような細かい攻略ではなく、スクリーンショットとコメントのみでお送りします*2。

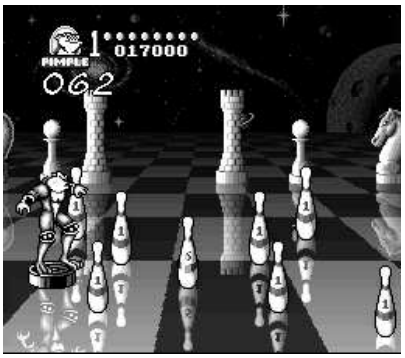
*2 面倒くさいわけじゃないよ。

◆ 1 人プレイの巻



横一列に並ぶピンを回収するピンプル（1P のキャラ）。
ここは楽勝。

上下二列に並んでいる所では、一人では全部回収できないので、どちらか片方のみに集中する必要があります。



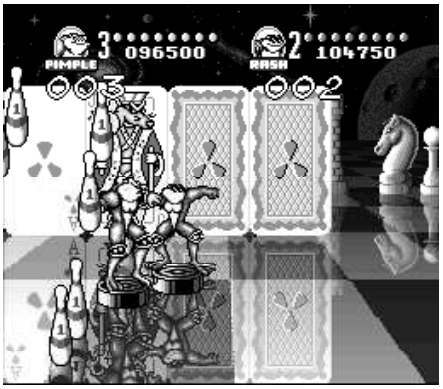
真ん中に見える 5 ポイントピンは絶対取るべし！

ドクロピンと、通常のピンが並んで設置されているところは、ドクロピンを取らないように注意すべし。

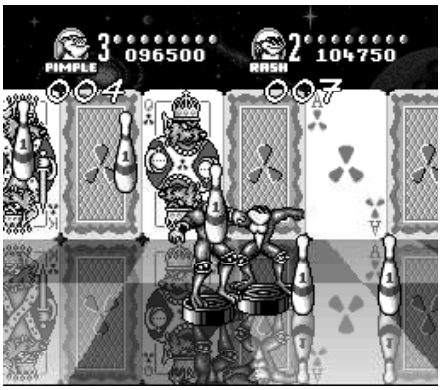
と、このように 1 人でゲームをしている限りでは、割と平和なステージです。
では、2 人でやるとどうなるか見てみましょう。

闘蛙

◆ 2 人プレイの巻

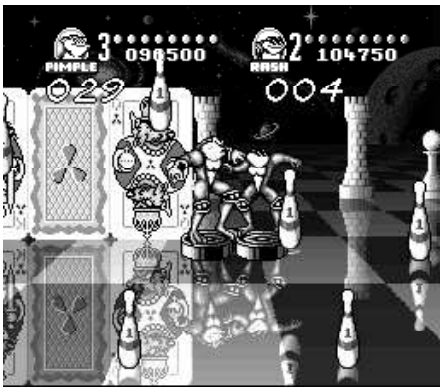


ピンを取り合うピンプル(1P、左のほう)とラッシュ(2P、右のほう)。
ﾋﾟﾝﾌﾟﾙ 「おい！邪魔すんな！！」
ﾗｯｼｭ「うるせえ！ピンよこせ！！」

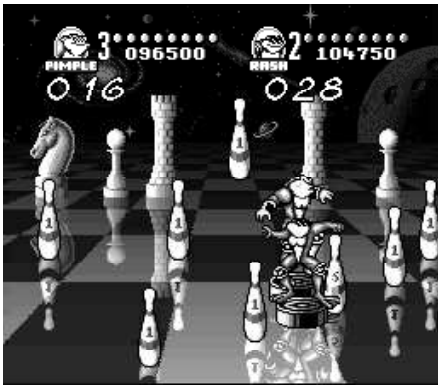


ピンが横一列に配置されているところだと、争奪戦がさらに激化。
ﾗｯｼｭ「ピンゲット！１！！」
ﾋﾟﾝﾌﾟﾙ 「ちょｗｗおまｗｗｗ邪魔ｗｗｗ」

ピンが 2 列に配置されているところでは、上下に分かれて、回収しましょう。
ﾗｯｼｭ「俺、上の段とるから、お前下の段な。」
ﾋﾟﾝﾌﾟﾙ 「おｋ」



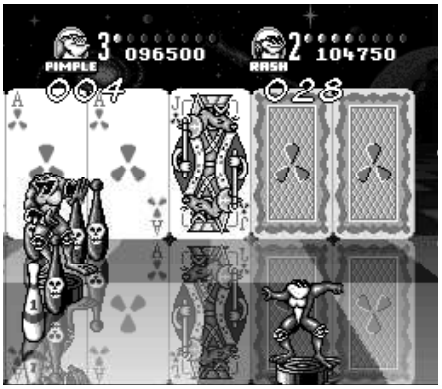
ラッシュの邪魔をするピンプルさん。
ﾗｯｼｭ「こっちくんなｗｗｗ」



5 ポイントのピンがあると、ほぼ確実に争奪戦が展開されます。

ﾋﾟﾝﾌﾟﾙ 「**うおおおおおおおおおお！！！**」

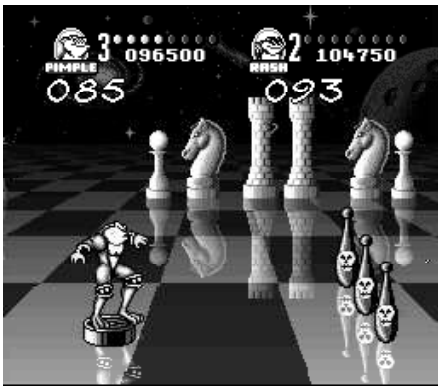
ﾗｯｼｭ「**うおおおおおおおおおお！！！**」



ピンの争奪戦に夢中になりすぎて、ドクロピンを取ってしまったピンプルさん。

ﾋﾟﾝﾌﾟﾙ 「ぐえっ！」

ﾗｯｼｭ「ざまあｗｗｗｗｗ」



ラッシュの HP が無くなったので、ずっとピンプルのターン！

この場合、ピンプルの HP が無くなるか、ゴールするまで、ラッシュはお休み。

■あとがき

攻略するとか言いつつ、全然攻略してない**何だかよく分からない**記事になってしまいましたが、お許し下さい。

次回は、**超鬼畜**な 4 面を攻略する予定です。お楽しみに！

入門 Ruby on Rails Vol.3

文 編集部 いのひろ

こんにちは

こんにちは。2 学期が始まりました。今学期もよろしくお願いします。

さて

この連載も 3 回目になりました。1, 2 回目では Rails アプリケーション開発の前提知識などをつらつらと書いてきましたが、やっとアプリケーション開発に着手してみたいと思います。

そこで、記事で扱うネタを考えたのですが、なかなか良いアプリケーションが思い浮かびませんでした。「twitter もどき」を作ってみるのも良いかなと思いましたが…。

既にご存知の方も多いかと思いますが、我らが情報（科）学類誌 WORD は 2009 年 1 月で、なんと創刊 30 周年を迎えました。9 月の末には記念イベントなどを企画していますが[*1]、そこで 30 年間の WORD の総集編を配布する事にしました。過去の原稿などをひっくりかえして編集作業にあたっていますが、これを良い機会にデータベース化したら良いのではないかという案がでたので、それ用の Web アプリケーションを開発しています。

それがパブリックになるかまだ決まっていませんが、Rails 製なのでこれが良いネタになるのではないかと考えたわけです…。

という事で簡単なアーカイブアプリケーションを作ってみようと思います。簡易図書館？です。

要求定義/仕様策定

今回のアーカイブアプリケーションに必要な機能は、「冊子情報の記録」「記事情報の記録」「編集者情報の記録」「PDF ファイルの保存」です。これらを考慮すると、今回のアプリケーションには以下のようなデータモデルが考えられます[*2]。

- 冊子情報（Book）
  - 冊子タイトル（title:string）
  - 発行年月日（pub_date:date）
  - 編集長情報（chief_author_id:integer）
- 記事情報（Article）
  - 記事タイトル（title:string）
  - （冊子の中での）開始ページ（start_page:integer）
  - （冊子の中での）終了ページ（end_page:integer）

*1 この記事が発行される頃には既に終わっているかもしれませんね

*2 各テーブルの主キーとなる「id 列」は Rails が勝手に挿入してくれるので考えなくても大丈夫です

・記事カテゴリ（category_id:integer）
・冊子情報（book_id:integer）
・編集者情報（author_id:integer）
・PDF ファイル情報（pdf_file_id:integer）

・編集者情報（Author）
・名前（name:string）
・ハンドルネーム（handle_name:string）
・誕生年月日（birth_day:date）
・WORD 加入年月日（enter_date:date）
・WORD 引退年月日（leave_date:date）

・カテゴリ情報（Category）
・カテゴリ名（name:string）

・PDF ファイル情報（Pdf_file）
・ファイルパス（path:string）
・名称（name:string）
・サイズ（size:integer）
・ファイルタイプ（file_type:string）（一応…

前回は Rails が RESTful であるか確認する為に、title と body を持つ Article というモデルを考え、scaffold[*1] を用いて簡単な登録／編集のページを生成しました。

今回はその Article モデルの拡張と、Book, Author, Category, Pdf_file を新たに追加します。しかし scaffold で既に生成されている View を変更するのはなかなか手間なので、新しいプロジェクトを作ってしまいましょう。

```
project$ rails archive
```

archive アプリケーションプロジェクトが生成されました。続いて、scaffold で足場を作ってしまいましょう。scaffold 生成のコマンドは、「script/generate scaffold model_name（singular）field_name:type 」の書式です。

```
project$ cd archive
archiv $ script/generate scaffold book title:string pub_date:date chief_author_id:integer
```

*1 Rails で“足場”を作るコマンド（script/generate scaffold）のことです。

```
archive$ script/generate scaffold article title:string start_page:integer end_page:integer
                                category_id:integer book_id:integer author_id:integer pdf_file:integer
archive$ script/generate scaffold author name:string handle_name:string
                                    birth_day:date enter_date:date leave_date:date
archive$ script/generate scaffold category name:string
archive$ script/generate scaffold pdf_file path:string name:string site:integer file_type:string
```

RDB に対して実際に CREATE TABLE などをするために、一度

```
archive$ rake db:migrate
```

を実行してください。これでデータベースにテーブルが作られました。

Model の変更

足場が出来たら、各 Model 間のリレーションシップを書き足します。例えば記事（Article）は冊子（Book）と主従関係になります。一つの冊子には複数の記事がありますよね。なので Article モデル、Book モデルの双方に主従関係のリレーションシップを書きます。

```
class Article < ActiveRecord::Base
        belongs_to :book
end
```

（app/models/article.rb）

```
class Book < ActiveRecord::Base
        has_many :articles
end
```

（app/models/book.rb）

Article には「belongs_to :book」（Book に属しますよ）という宣言、Book には「has_many :articles」（いくつかの Article[*1] を持ちますよ）という宣言を書きました。

これを書くと何が嬉しいのでしょうか。例えば任意の Book から、それに属する Article の情報

*1「いくつかの Article」なので「has_many **:articles**」（複数形）になっています。気をつけてください。

を取得する事が出来ます[*1]（下の例で利用するコードは後ほど説明するので安心してください）。

```
book = Book.find_by_id（ 1 ）# id が 1 の Book を RDB から取得して変数 book に代入
book.articles                    # book_id が 1 である Article（＝ book に収録された記事）の一覧
```

逆に任意の Article の book_id を見て、その Book の情報をとってきてくれます。

```
article = Article.find_by_id（ 1 ）# id が 1 の Article を RDB から取得して変数 article に代入
article.book                     # article.book_id の Book
```

さて、同じように別のテーブル間も定義します。記事（Article）はただ一つの記事カテゴリ（Category）を持つため、「has_one」を使います。

```
class Article < ActiveRecord::Base
        belongs_to :book
        has_one :category
end
```

（app/models/article.rb）

```
class Category < ActiveRecord::Base
        belongs_to :article
end
```

（app/models/category.rb）

*1 もちろんデータが既にある場合を考えてください。

さらに記事（article）はただ一つの PDF ファイル（pdf_file）を持つので、同様に以下のように書き足します。

```
class Article < ActiveRecord::Base
        belongs_to :book
        has_one :category
        has_one :pdf_file
end
```

（app/models/article.rb）

```
class Pdf_file < ActiveRecord::Base
        belongs_to :article
end
```

（app/models/pdf_file.rb）

これでテーブル間のリレーションシップが完了しました。とりあえず実行してみましょう。

```
archive$ script/server
```

ブラウザで「http://localhost:3000/articles/new」にアクセスします。

適当に「/authors/new」や「/books/new」「/categories/new」などにアクセスして適当に登録しておくといいかもしれません。

Controller の変更

これでフォームからデータを登録する事が出来ますが、これだとデータ型が Integer であるフィールドの入力が多少手間です。何が言いたいのかというと、例えば記事（Article）の編集者（Author）のデータ型が Integer 型であるため、ブラウザのフォームからは数字を入力しないといけません。しかし Author テーブルに一度問い合わせをすればフォームに編集者一覧を列挙することができます。名前で選択なら簡単だね！という風にしようというわけです。

それには必要なデータを articles_controller でロードし、さらに新規 Article 追加の View を、そのデータを利用してドロップダウンリストを表示するように書き換えます。

まず app/controllers/articles_controller.rb を編集しましょう。ファイルを適当なテキストエディタで開き、new メソッドを見つけてください。「@article = Article.new」のあたりに「@book=~」「@author=~」「@category=~」の 3 行を追加します。

```
def new
        @article = Article.new
        @book = Book.find( :all, :order => "id" )
        @author = Author.find( :all, :order => "id" )
        @category = Category.find( :all, :order => "id" )
        .....
```

（app/controllers/articles_controller.rb）

これらはデータベースに格納されているすべての Book, Author, Category のレコードをハッシュとして取得し、「@なんたら」というインスタンス変数に格納しています[*1]。「:order => "id"」は id 列で昇順に取得するという事です。他の 2 つも同じ事をやっています。

find メソッド

ここで利用している「find」というメソッドは ActiveRecord[*2] のクラスメソッドで、条件などを指定してデータベースからモデルを引っ張ってくるときによく利用します。

たとえば id 列の値が 1 のレコードを取得する場合は、

```
book = Book.find( :all, :condition => [ 'id=1' ] )
```

や、

*1 ここでの「Book」は Book モデルのことで、モデルに対する操作は ActiveRecord が SQL に変換して実行してくれます

*2Ruby の Object Relational Mapping ライブラリ

```
book = Book.find_by_id( 1 )
```

と書いたりします。これらはどちらも同じオブジェクトが変数 book に代入されますが、該当するオブジェクト（ここでは id が 1 であるレコード）が見つからなかった時の挙動が異なります。

「find(:all, :condition)」で検索した場合、該当するレコードがデータベース上に見つからなかったときは「例外」が発生します。なので「begin ~ rescue ~ end」などで例外を捕捉しないと何かあったときに大変なことになります。

また「find_by_hoge()」（hoge は任意のカラム名）で検索した場合では、「nil」が変数 book に代入されます。なので「if book.nil?」など nil の判定だけで済むので楽です。おすすめです。

さらに「find_by_hoge()」は単一カラムだけでなく複数のカラムに条件をつけて検索することもできます。そのような場合には「find_by_hoge_and_fuga(hoge, fuga)」と書いたりします。

```
book = Book.find_by_id_and_title( 1, "WORD 11 号" )
# 「id=1」かつ「title=WORD11 号」なレコードを探して変数 book に代入する
```

これを「find(:all, :condition)」でやる場合は、

```
book = Book.find( :all, :conditions => [ 'id=1', 'title=WORD11 号' ] )
# 「id=1」かつ「title=WORD11 号」なすべてのレコードを探して変数 book に代入する
```

個人的には find_by_hoge の書き方がおすすめです。

find は非常に便利で強力なメソッドです。RDB に対する SELECT 文などは、この find が担当します。つまり SQL における ORDER BY や TOP などの指定も可能です。つまりは、適当に書くとデータベースに沢山の SQL が飛んだり、大変おもたい処理をさせてボトルネックになるという危険性も含んでいます。気になる人は調べてみてください。

script/console の利用

find メソッドを実行したときに、RDB からどのようなデータが取得できているかというような確認には「script/console」を用いると便利です。プロジェクトのルートディレクトリに移動し、

```
archive$ script/console
```

とやってみてください。

```
Loading development environment（Rails 2.1.0）
>>
```

というような irb（Interactive　Ruby）のようなものが起動したでしょうか？ほとんど irb と同じものなのですが、「script/server」をやったときと同じように Rails の実行に必要なライブラリや「conf/」以下の設定ファイル[*1] を読み込んでいるため、ActiveRecord などの拡張ライブラリの require などをいちいち行う必要はありません。

さて、コンソールが起動したら実際に先ほどのコード[*2] を試してみましょう。

```
>> Book.find_by_id（ 1 ）
=> #<Book id: 1, title: " q あ w s で r f g t ひゅじこ l p ; ", pub_date: "2009-09-14", created_at:
"2009-09-14 12:03:04", updated_at: "2009-09-14 12:03:04", chied_author_id: 1>
>> Book.find（ :all, :conditions => [ 'id=1' ] ）
=> [#<Book id: 1, title: " q あ w s で r f g t ひゅじこ l p ; ", pub_date: "2009-09-14", created_at:
"2009-09-14 12:03:04", updated_at: "2009-09-14 12:03:04", chied_author_id: 1>]
```

（すでにデータが入っていれば）このように RDB からレコードを取得できていることがわかります。今回の記事の前半にテーブル間のリレーションシップを定義すると何がうれしいか、の例として書いたコードも試してみてください。その他、いろいろ手軽に試す場所として script/console を使うとよいと思います。

View における Ruby コードの埋め込み

さて、次に app/views/articles/new.html.erb を変更します。「erb」ファイルでは Html に Ruby のコードを埋め込む事が出来ます。

View における Ruby コードの埋め込み方法は大きく分けて２つあります。一つは HTML 出力に直接的に関与するもの、もう一つは間接的に関与するものです。

HTML 出力に直接的に関与するというのは、その Ruby コードが実行されたときに結果がそのまま HTML として出力されます。たとえば、

```
<%= link_to 'say', 'hello/world' %>
```

*1 おもに定数などを書いたりします

*2「Book.find（ :all, :condition => { 'id=1' } ）」もしくは「Book.find_by_id（ 1 ）」など

といったようなハイパーリンクを生成するメソッドを使うと、

```
<a href="hello/world">say</a>
```

と出力されます。この場合コードを埋め込むときは「<%=」と「%>」で囲みます。
　また間接的に関与するというのは、ループ文などを埋め込むことができると言うことです。たとえば、

```
<% 5.times do %>
        <%= link_to 'say', 'hello/world' %>
<% end %>
```

などは、

```
<a href="hello/world">say</a>
<a href="hello/world">say</a>
<a href="hello/world">say</a>
<a href="hello/world">say</a>
<a href="hello/world">say</a>
```

を生成します[*1]。コードは「<%」と「%>」で囲みます。先ほどと異なるので注意してください。

View の変更

　さて本題に戻ります。View では先ほどコントローラ側でロードしておいたデータ（それぞれ @book, @author, @category）を用いて Book、 Author、Category の選択をしやすくします。app/views/articles/new.html.erb を編集します。

```
<%= f.text_field :author_id %>
```

（app/views/articles/new.html.erb）

という行を見つけてください。これは実際にブラウザで表示したときに、記事（Article）の編集者（Author）の id（Integer）を入力するテキストボックスを生成するためのコードです。このま

*1 もちろん、「do」を「{」、「end」を「}」とすることもできます。

まだとわざわざ編集者とその id の関連づけを人間が覚えていないといけないので、使い勝手がよくありません。これを

```
<%= collection_select（ 'article', 'author_id', @author, 'id', 'name' ） %>
```

（app/views/articles/new.html.erb）

と書き換えましょう。「collection_select」というのはドロップダウンリストを生成するものです。各引数は順番に

object_name:このフォームでデータを登録するモデル
field_name: 第 1 引数で指定したモデルのフィールド名
list_of_objects: ドロップダウンリストに表示するオブジェクトのリスト
value_field: ドロップダウンリストから実際に取得されるフィールド
display_field: ドロップダウンリストに列挙されるフィールド

となっています。ちょっとわかりにくいですが、ここでは「@author から取得したデータの中から name フィールドをドロップダウンリストに列挙し、選択されたものの id フィールドを Articles テーブルの author_id に割り当てるよ」という感じです。

同じように category と book のところも以下のように書き換えましょう。

```
<%= collection_select（ 'article', 'category_id', @category, 'id', 'name' ） %>
<%= collection_select（ 'article', 'book_id', @book, 'id', 'title' ） %>
```

ブラウザで変更を確認してみましょう。

```
archive$ script/server
```

Author, Category, Book のフォームがドロップダウンになってますね！

とりあえず今回はこんなところにしておきましょう。Controller と View をいじってみていかがでしたか？ Ruby on Rails では、このように Controller と View、そして Model を上手に役割分担させて Web アプリケーションを実現しているのが少しわかったのではないかと思います。

実は今回の変更だけでは「/articles/new」で新しく記事情報を追加することができません。PDF_file の情報がテキストボックスからだと正常に入力できない為です。これを解決するためには Author や Book と同じように、Controller でデータのロード、View で collections_select の利用をすることで解決できます。是非変更して新しい Article の追加ができるようにしてみてください。

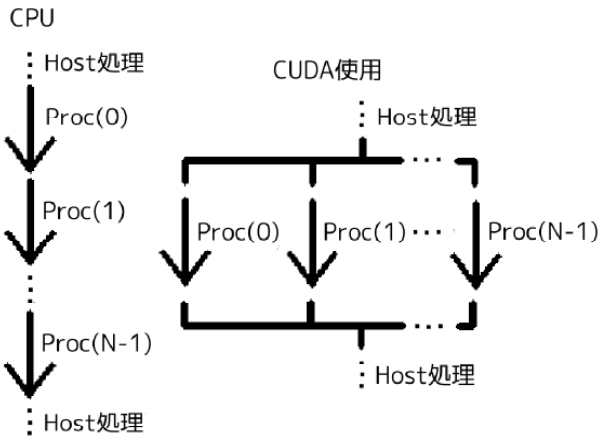
CUDA入門

文 編集部 Flast

皆さんは CUDA というものを知っているでしょうか。知っていても Adobe の製品の一部の処理が早くなるとか、エンコーダによっては早くなってくれたりとか、そのような程度ではないでしょうか。ここでは CUDA がどういうものなのか説明[*1] しながら、少し CUDA でのプログラミングも解説したいと思います。

CUDAとは

CUDA[*2] とは Compute Unified Device Architecture の略でクーダと読みます。CUDA は NVIDIA 社の GPU[*3] を使ってプログラミングをするためのコンパイラやライブラリを集めた統合開発環境のことを指します。主な開発言語は C 言語ですが、その他の言語からでも呼び出せるバインディングが公開されています。



図：CPU と CUDA の実行フロー

CUDA を用いた場合の演算速度ですが、使うアルゴリズムによって効率や処理速度がかなり変化します。なぜなら GPU の実行形態が超並列といわれる方法だからです。右の図をみてもらうとわかりやすいと思いますが、CPU ならループ構文を使って何回も繰り返す処理を CUDA では大量のコアが同時に処理を行います。同時に実行されるということが非常に重要で、漸化式のような前の演算結果が必要なアルゴリズムでは、それほど性能向上は見込めないということです。しかし CUDA の得意な行列演算などの場合、コードのチューニング次第ですが CPU のループを用いた場合よりも 3 ～ 400 倍程度の性能向上が見込める[*4] そうです。

どれほどの数のコアが同時に実行するか比較してみると、最近の CPU では 2 個（デュアルコア）や 4 個（クワッドコア）が主流ですが、GPU によってかなり差があるものの最低 8 個や一般ユーザー向けでは 64 個や 128 個、研究目的に使われるようなものでは最大 960 個と莫大な数のコアが同時に処理を実行します。

---

*1 この記事では CUDA の細かいインストール・設定などは説明しません。あくまで CUDA とはどのようなものなのかをお伝えするだけです。

*2http://www.nividia.co.jp/object/cuda_home_jp.html

*3 すべての GPU で使用可能なわけではありません。使用可能な GPU のリストは http://www.nvidia.co.jp/object/cuda_learn_products_jp.html を参照してください。

*4http://journal.mycom.co.jp/special/2008/cuda/008.html

CUDA 入門

Hello, CUDA!

CUDA でのプログラミングを少し紹介したいと思います。詳細な仕様やプログラミングの方法は NVIDIA が公開しているマニュアル[*1] を参照してください。

今回は N 次元ベクトル A, B を加算して結果を C に代入するという非常に簡単なプログラムを C 言語版と CUDA 版の 2 通りで紹介したいと思います。以下のサンプルコードでは適当な定数 N が定義され、A, B には既に値が代入されているものとします。

まず C 言語版ですが、これは for ループで単純に各要素を加算している簡単な方法です。

```
void main( void )
{
    float A[ N ], B[ N ], C[ N ];
    int i;

    for ( i = 0; i < N; ++i )
    {
        C[ i ] = A[ i ] + B[ i ];
    }
}
```

次に CUDA 版での場合ですが、演算する GPU が参照するメモリは VRAM なので、CPU がアクセスする RAM から VRAM へと転送する処理が必要になります。細かいことは置いておいて先に CUDA 版のサンプルコードを見てください。

```
__global__ void VecAdd( float *A, float *B, float *C )
{
    int i = threadIdx.x;
    C[ i ] = A[ i ] + B[ i ];
}

void main( void )
{
    float A[ N ], B[ N ], C[ N ];
    float *d_A, *d_B, *d_C;
    int size = sizeof( float ) * N;

    cudaMalloc( ( void ** )&d_A, size );
```

*1http://www.nvidia.co.jp/object/cuda_develop_jp.html

```
    cudaMalloc( ( void ** )&d_B, size );
    cudaMalloc( ( void ** )&d_C, size );

    cudaMemcpy( d_A, A, size, cudaMemcpyHostToDevice );
    cudaMemcpy( d_B, B, size, cudaMemcpyHostToDevice );

    VecAdd<<< 1, N >>>( d_A, d_B, d_C );

    cudaMemcpy( C, d_C, size, cudaMemcpyDeviceToHost );

    cudaFree( d_A );
    cudaFree( d_B );
    cudaFree( d_C );
}
```

まず注目してもらいたいのは、CUDA 版のサンプルコードには一切ループ構文が存在しないことです。これは CUDA では GPU の 1 つのコアが実行する関数を記述するだけでそれを大量のコアで実行するように適応することができるからです。CUDA でのプログラミングがわかりやすい点はここにあります。それではサンプルコードの解説をしていきたいと思います。

まず CUDA では CPU 側を Host、GPU 側を Device と言います。Device 側のメモリを確保する為には C の malloc()や C++の new ではいけません。これらの方法だと Host 側のメモリが確保されるだけです。Device 側のメモリを確保するには cudaMalloc()を使います。

使い方は第 1 引数に確保したい変数へのポインタのポインタを渡します。第 2 引数には確保したいメモリサイズをバイト単位で指定します。戻り値はエラーコードなので間違えないようにしてください。また、確保した領域を確保するためには cudaFree()を使います。

Host から Device へ、Device から Host へデータを転送する時はどちらも cudaMemcpy()を使います。第 1 引数には転送先、第 2 引数には転送元を指定し、第 3 引数には転送するサイズをバイト単位で指定します。この関数の第 4 引数に cudaMemcpyHostToDevice を指定した場合は、Host 側の RAM から Device 側の VRAM へと転送され、cudaMemcpyDeviceToHost を指定した場合は Device 側の VRAM から Host 側の RAM へと転送されます。

そして目新しい変化といえば VecAdd()についている__global__と threadIdx や、VecAdd()を呼び出すときの<<< >>>でしょうか。これは CUDA で追加された構文です。

まず__global__は、これが Host 側から呼び出せる GPU 上で動作する関数であることを指定します。これが指定されていないと CUDA のコンパイラである nvcc は、デフォルトで Host 上で実行される関数だと認識してしまいます。なので__global__の指定は忘れないようにしましょう。

__global__以外にも Device で実行される関数が呼び出す関数の為の__device__というのもあります。__device__が指定された関数は Host 側からは呼べません。

次に threadIdx ですが、現在実行されているスレッドがどのスレッドかを識別するための変数です。この使い方だと threadIdx.x 番目のスレッドが各ベクトルの threadIdx.x 番目の要素同士を加算しています。

今はスレッドの分割が 1 次元上で行われているので threadIdx の x 要素しか用いていませんが、

最大 3 次元まで分割することができます。

そして<<< >>>ですが、これは先ほどの__global__を指定した関数を呼び出す時に使うためのものです。この 3 重山括弧の中の値によって、どのように Device 上で実行するかを指定することができます。サンプルの<<< 1, N >>>では N 個のスレッドが実行されるように指定しています。この値でスレッドの分割を行うことができます。

今は 1 次元で分割しているため、整数を指定していますが、N × N の 2 次元で分割しようとした場合、dim3 型の変数を用いることで 2 次元は

```
Proc<<< 1, dim3( N, N ) >>>( … );
```

と表すことができ、N × N × N の 3 次元で分割しようとした場合、

```
Proc<<< 1, dim3( N, N, N ) >>>( … );
```

と書くことができます。これらの方法で実行する場合__global__関数側で threadIdx の y 要素や z 要素を使って正しくスレッドの区別する必要があります。

これら最大 3 次元で分割されるスレッドの集合を CUDA では Thread Block といいます。CUDA ではさらにこの Thread Block の集合である Grid という概念などがあります。スレッドがどのように実行されるかを指定できるほか、メモリの使い方もある程度指定できます。

おわりに

CUDA について軽く説明してみました。CUDA はこの紙面上だけで説明しきれないほど様々なことができます。しかし、残念なことに現在 CUDA を使ったソフトウェアはそれほど多くありません。それは一般用途ではあまり使用する機会がないからなのではないでしょうか。

一方、研究用途では CPU だけで構成された大型のクラスタを使っていたりしますが、まだ処理能力が足りないという分野は非常に多くあります。物理演算などはよく並列化の議題にあがる分野でもあります。もし、自分の研究がこういった分野で、CUDA を使えそうなら使ってみてください。CUDA のプログラミングの容易さと処理能力はあなたに衝撃を与えるでしょう。

編集後記

情報科学類誌

WORD

From College of Information Science

WORD 政権交代号

発行者　情報科学類長

編集長　柴田 泰晴

製作・編集　筑波大学情報学群
情報科学類 WORD 編集部
（第三エリア C 棟 212 号室）

2009 年 9 月　初版第一刷発行