Digital ESSAY

「rust でがんばる」

坂口 裕靖

監視カメラ案件ですが、日々ちょっとづつ進化しております。当初 perl にてサムネイルを生成していたのですが、いかんせん遅い。どげんかせんといかんということで、一念発起して別な言語で実装する方針にしました。

根本的な問題としては perl で遅いというところにあるわけですから、インタプリタ系の言語は多分難しいんじゃないでしょうか。というわけで、コンパイラ系の言語を探すことになります。これがいろんなマシンで動かす必要があるとなると大変悩ましい問題なわけですが、とりあえず今使ってるサーバで動けば十分なので、その意味では互換性とかをあんまり気にする必要がないのはいいところです。まあ順当には C++ とかでやればいいんでしょうけど … えーと … 単にサムネイルを生成したいという、やりたいことに対してちょっと学習コストが重すぎるよな、という感じでイマイチやる気が出ません。とりあえず評判良さそうだということで、rust で実装してみることにしました。

まあしかし、これが結構ややこしかった … ガベージコレクション不要という美点を持たせるために、基本的にポインタの共有を許さないという設計思想。ポインタは単一のスコープが「所有権」として所持していて、スコープから外れると廃棄されるという事になってます。で、ポインタを「借りる」ことはできるのですが、借りた方のスコープは所有権を持っていないため、スコープから外れても実体データは破棄されません。そして、ポインタを借りない場合は自動的に所有権が移動します。ということは、for ループの外側で定義したオブジェクトに for ループの内側で借りずにアクセスすると、所有権はそのスコープに移動され、したがって最初の for ループが終わると、そのオブジェクトは破棄されることになります。で、話をややこしくするのが、この所有権移動規則が適用されるのはヒープに確保したオブジェクトについてのみ、ということです。スタックに確保されるような変数は、移動しません。まあとにかく、今使おうとしているオブジェクトはスタック変数なのかヒープ変数なのか、それをどこで作ってどこで破棄するか、を常に意識しながらじゃないと使えないというのが、なかなか慣れないところでした。なにしろ設計のセントラルドグマなので、この部分を乗り越えないと 1 ビットたりとも動かないのです。

もう一つの慣れなさが、変数はデフォルトでは変更不能である、というところです。フツーに考えれば定数と変数があると、変数は変更可能だと思うじゃないですか。違

One Point BUZZ WORD

差分検知

ちゃんとしたアルゴリズムはあるんでしょうけど、とりあえず今使ってそれなりに見えるのはこんな感じ。前提としてカメラは固定とし、motion で収録するものとします。そもそも処理する画像のサイズですが、原解像度の必要はないです。視力と相談してなるべく小さくすると便利。今の所横 640 でやってますが、あまり困ってません。

基準フレームと現在のフレームでピクセルごとに比較して、差分がなければ黒、差分があるならその色で塗るとします。本当は RGB 全部でチェックするといいのですが、まあ G チャンだけ比較しても大体問題ないっす。そいでもって、カメラにはノイズがつきものなので、適当な閾値を決めて、差分の絶対値が閾値以下なら差分なしとした方が良いようです。実際の値はカメラとか環境とかによるのでしょうが、とりあえず 32 で回してます。

あとは基準フレームをどこに置くかなのですが、基本的には撮影イベントの最初のフレームで良いようです。一つ前のフレームとの差分を取ると、「消えた分」「出た分」の両方が検出されるため、今のフレームで出た分は次のフレームで消えることになり、上書きしていくと違いがよくわからなくなります。これが最初のフレームと比較すれば、出た分だけが検出できるので安定します。一方で雲が太陽を隠すなど、急な照明条件の変化で大きな領域を差分として検出してしまうと、最終的には変化が見づらくなります。そこで、適当な閾値を決めて、変化があるピクセル数が閾値を超えた場合、直前のフレームとの差分を取るようにすると、割と見やすくなるようです。こちらも被写体で閾値は異なるのでしょうが、画面の 1/4 ぐらいで設定しています。どちらのアルゴリズムで検出したかがわかるように、最初のフレームとの差分と直前フレームとの差分で色を変えておくと、修正の方向性が見やすくなるかと思います。

うんだな、これが。変更可能な変数は mut というキーワードを付けて宣言しなければなりません。とすると、変数にはぱっと見て（1）不変なスタック変数、（2）可変なスタック変数、（3）不変なヒープ変数、（4）可変なヒープ変数、という 4 種類があることになります。実際にはこれに加えて（5）借りてきた不変なヒープ変数を不変に使用、（6）借りてきた可変なヒープ変数を不変で使用、（7）借りてきた可変なヒープ変数を可変で使用、とかスマートポインタとか色々あるようですが、とにかくデフォルトでは不変です。不変の何がいいかというと、書き戻されないことが保証されているので、コンテクストを気にせず同じ値が使えるということなんだろうと思います。一方で可変な変数については、どのタイミングで読み出すかによって値が異なるため、処理の順番を入れ替える場合は注意が必要です。また、所有権規則として「同時に 2 箇所以上に可変な貸し出しはしない」「不変で借りてきた所有権は移動できない」というのがあり、大変繊細なものになっていてややこしいです。これが脊髄反射的に状況が理解できるようになれば良いんでしょうが、なかなか ...

一方でコンパイラが出すメッセージは詳細かつ的確なので、かなり助けられます。まあ大抵のポカミスはこの所有権絡みなので、はいはい借りまひょ、ありゃ駄目？じゃ可変にしときますわ、とかやってると自動的に問題がないプログラムができあがるという寸法なのでした。結果としてメモリ管理上問題がないなら、いいんじゃないかな。うん。なんかこの、言語習得初期における処理系におんぶでだっこな期間って、右も左もわからない具合がなんとも楽しいですよね。

でまあ所有権規則の魔宮をくぐり抜けたとして、今度は実行環境との相互作用という壁があります。ディレクトリを取得するにはどうするか、ファイルリストを取得するにはどうするか、ファイルを開くにはどうするか、ファイルに書き出すにはどうするか、外部コマンドを実行させるにはどうするなどなど。でこの山を超えると同時に、rust のエラーハンドリングという、第二の魔宮が待ち構えているわけです。rust は構造化エラー処理を持たない代わりに、エラーが発生しうる場合、enum を返します。rust の enum は tuple みたいにいろんな型を持てるので、「成功した場合の型」と「失敗した場合の型」などを含む enum を処理する必要があります。真当に対処するなら、すべてのありうる型との一致を調べて適切な処理をするわけで、しかも抜けがあるとコンパイラがビシビシ指摘してくるわけですが、まあそこまで頑張らなくてもいいんじゃないの、という場合もあります。というか、趣味でなんかやってる場合はだいたいエラーなら死んでおけ、なわけです。それ以前に動かないんだよ〜ってわけで。rust の場合、素晴らしいことにこのズボラ処理用の syntax sugar が山ほど用意されてることで、「エラー処理しなきゃいけないよね、うん、わかる。正しい。わかってるんだけどね、無視したいんだ、俺は。とりあえずこいつは。頼むから頓死させてくれ」という心境を表す expect（）とか unwrap（）とか unwrap_or（）とかが用意されており、エラーを華麗にスルー（というか頓死）できるわけです。まあ rust の哲学からすると、回復可能なエラーは回復しとけやボケ、って感じだろうと思うので、可能であればそちらに従うほうが良いでしょう。特にライブラリを公開することを考えているなら、このあたりをよく考える必要がありそうです。まあでも、エラー対処はどうしても状況依存になるため近視眼的なコードがずらずら続くことになり、ソースの見通しが悪くなるのはちょっといやーんな感じ。

そんなこんなでコンパイラと二人三脚（というか二人羽織）でなんとかアルゴリズムを実装してみたところ、perl だと動画 1 本につき数十秒かかっていた処理が、数秒程度にまで高速化できました。動画の処理は餅屋にまかせて ffmpeg で ppm フォーマットで書き出してもらい、こいつらを rust で処理して ppm のサムネイルを書き出し、ImageMagick の convert で jpg に変換して書き戻し、中間ファイル群を削除するというような処理です。実行するマシンのメモリはさほど潤沢ではないため、バッファをちまちま使いながら処理するわけですが、ガベージコレクタが動作しないため、最初から最後までほぼ同一ペースで処理できるのは頼もしいところです。C# とかだと頻繁にガベージコレクタを起動して、不要だけど捨ててないバッファを回収しないとメモリリークが発生するところですが、この苦労をコンパイル前に、人間様の方で負担しておく、というわけですね。今回ちょっと手応えを感じたので、もうちょっと色々実装してみようかなと考えてます。今更 perl の手軽さを捨てる気もないのですが、とっつきにくいけど rust は結構いいかもしれません。まあちょっと go も味見してみて、決めたいところでもありますが

Hiroyasu Sakaguchi
株式会社 IMAGICA Lab.