PC 4K Intro minify テクニック

BY よっしん

2021/12/11　TOKYO DEMO FEST 2021

2021/12/12　若干の加筆修正

自己紹介

- よっしんです
  - https://yosshin4004.github.io/index.html
  - https://twitter.com/yosshin4004
- ゲーム開発系
- 0x4015 という捨て垢（のようなもの）でデモを作っていました

概要

デモ・デモパーティ・デモシーンとは

- デモ
  - 技術力を誇示して競うことを目的として作成される、音と映像をリアルタイムに生成するコンピュータ・プログラム。
  - 日本では「メガデモ」という呼称が一般的。
- デモパーティ
  - デモの発表の場として開催されるイベント。
  - デモが大スクリーン・大音量で上映され、観客の投票で優劣が決定される。
- デモシーン
  - デモ・デモパーティおよびその背景を含む文化の総称。

PC 4K Intro とは

- デモのカテゴリの一つ
  - ファイルサイズ 4095 バイト以内の実行ファイルで、外部のデータに頼ることなく（opengl32.dll など、OS 標準搭載されている dll は利用可能）、プロシージャル技術などを駆使しながら、音と映像を作りだすことが求められる。
- 難易度の高いカテゴリ
  - ファイルサイズ制約から、既存ライブラリやゲームエンジンなどの活用はできない。
- だが近年はお手軽カテゴリ化
  - 攻略されまくった結果、ほとんど既存テクニックと既存ツールの活用で作成できる状況。
  - あとはファイルサイズ上限だけを気にしていれば良い。

ファイルサイズ制約の存在意義

- 作成者のスキルのみで優劣が決まる土俵を作り出すこと
  - 既存ソフトウェアの再利用を困難にする。
  - アセットの物量勝負になることを避ける。
- 先人たちと同じ土俵で競える条件を提示すること
  - このカテゴリの難攻不落とされる最強デモ群には、十年以上前の作品が多く存在。
  - それらとほぼ同条件で競うことができる。
    - GPUの進化による恩恵は受けられるがそれ以外の条件はほぼ同じ。
- ファイルサイズ制約は必要悪のようなもの
  - 制約が存在することにより、フェアな競争が可能になる。

PC 4K Intro の作り方

- レンダリング周りはシェーダをフル活用
- サウンド周り
  - PC 4K Intro に特化したドライバが利用可能
    - https://github.com/hzdgopher/4klang
    - https://github.com/askeksa/Oidos
  - もしくはサウンドもシェーダで作る
- シェーダコードは、Shader minifier （後述）で minify する
  - minify = サイズを最小化すること
- 実行ファイルは exe パッカーの crinkler（後述）を使って作成

本資料について

- PC 4K Intro の作成手法のうち、minify に特化した内容をまとめていきます。
- Windows exe に限定した内容を多く含みます。
- 30分しかないので、かなり駆け足になります。ご了承ください。

Shader minifier

Shader minifier とは

- PC 4K Intro などで使われる定番の minifier
  - by Laurent Le Brun (LLB / Ctrl-Alt-Test).
- シェーダコードを、動作を変えずに短くするツール。
- Github から入手可能
  - ソースコード
    - https://github.com/laurentlb/Shader_Minifier
  - ビルド済みバイナリ
    - https://github.com/laurentlb/Shader_Minifier/releases

Shader minifier の制限

- シェーダの書き方によっては動作しない
  - マクロや構造体の使用に制限がある
  - 許可されていない構文がある
- コンピュートシェーダに対応できない
  - フラグメントシェーダと見せかけて minify させたのち、コンピュートシェーダの呼出し部分を後付けすれば使えなくもない。

Crinkler

crinkler とは

- PC 4K Intro などで使われる定番の exe パッカー
  - by Rune L. H. Stubbe (Mentor/TBC) and Aske Simon Christensen (Blueberry/Loonies)
- 既存の exe を圧縮するものではない
  - vc++ のリンカとして動作し、API 呼出しの仕組みを置き換えながらリンクを行い、圧縮された exe を直接生成。
- Github から入手可能
  - ソースコード
    - https://github.com/runestubbe/Crinkler
  - ビルド済みバイナリ
    - https://github.com/runestubbe/Crinkler/releases

crinkler で高い圧縮率を達成するには

- 高い圧縮率が得られる条件
  - 似たバイト値が近くに再出現する
  - 完全一致する長いバイト列が存在する
  - バイト値のバリエーションが少ない

- 以下のようなアプローチで圧縮率を上げられる
  - 副作用の無い命令の順序を入れ替え、似た命令を連続させる
  - なるべく既出の命令を使う

crinkler で圧縮された exe が起動する流れ

1. exe がメモリ上に展開される

2. _Import と呼ばれる処理を実行
   - crinkler が自動的に付加する 120 バイトほどのコード
   - opengl32.dll や user32.dll 等々に含まれる API のアドレスを取得し、ユーザープログラムから利用可能な状態にする
     - dll 内の API 名一覧は、文字列でなく 4 バイトハッシュ値で exe 内に格納することでサイズ削減。

3. ユーザープログラムを実行
   - _entrypoint というシンボルから開始

crinkler のオプション指定

- 推奨オプション
  - /REPORT
  - /COMPMODE:SLOW
  - /HASHSIZE:300
  - /HASHTRIES:300
  - /ORDERTRIES:300
  - /UNSAFEIMPORT
  - /OVERRIDEALIGNMENTS
  - /UNALIGNCODE
  - /NOINITIALIZERS

- 各オプションの意味はドキュメントを参照してください。

crinkler の注意を要するオプション

- /TINYIMPORT は非推奨
  - 巧妙な方法で dll 内の API アドレス取得を行うことで、より小さな exe ファイルを出力できる。
    - API 名から作成したハッシュ値をインデクスとし、全 API アドレスを関数テーブル上に展開。
    - ユーザープログラムから関数コールする時は、関数テーブル上の要素をダイレクト指定。
    - 結果的に、ハッシュ値そのものは exe ファイル上に出現しないこととなり、サイズが小さくなる。
  - しかしハッシュ値のコリジョン耐性が極端に低くなる。
  - windows update などにより、OS 側の dll 内に API 追加が発生すると、コリジョンが起きやすい。

- /TINYIMPORT は、将来の互換性の観点から利用は推奨されていない。
  - どうしても使いたい場合は、/TINYIMPORT を使わない exe も同梱することが推奨されているが、チート利用宣言のような認識であり、デモの評価上はマイナス要因。

手作業による人力 minify

最終手段：人力 minify

- Shader minifier と crinkler で、一発で 4095 バイトに収まればラッキー。
- しかしなかなかそうは行かず、かなりの時間を minify 作業に投入することになる。
  - ツールに頼れず、基本的には手作業となる。
- ファイルサイズを 0.1 ビットでも良いから削りたい、といったような状況が発生する。
  - サイズ削減のためには手段を選んでいられない。
  - 苦行

Size coding / code golf

- 最小の命令数で目標課題を達成するプログラミング
- これらはデモシーン以外の領域でも、プログラミングパズルや競技プログラミングの題材として存在する概念
- Size coding や code golf で検索すると、様々な minify テクニックを見つけることができる。

crinkler のレポートが重要な道しるべ

- /REPORT オプションを使うと生成されるレポートが便利
  - 圧縮元データの各バイトが、何ビットに圧縮されたのかを詳細に知ることができる。
  - これを観察しながら地道に minify していく
- _Import 処理の詳細も、レポート上で確認可能

Crinkler compression report

Report for file C:¥usr¥dev_area¥demoscene¥4k_demo¥mini4k¥AsmAsmRelease¥mini4k.exe generated by Crinkler 2.2 on Fri Apr 10 22:43:09 2020

Options: /SUBSYSTEM:WINDOWS /ENTRY:entrypoint /COMPMODE:SLOW /HASHSIZE:300 /HASHTRIES:300 /ORDERTRIES:300 /UNSAFEIMPORT /OVERRIDEALIGNMENTS /UNALIGNCODE /NOINITIALIZERS

Output file size: 964

Bits per byte:

collapsed sections globals sections + globals globals expanded sections + globals expanded

Address				Label name	Size	Comp. size	Ratio
-00420000				Code sections	523	275.18	52.6%
+00420000				_Import	122	104.15	85.4%
+0042007A				_entrypoint@0	115	47.19	41.0%
+004200ED				_GetProcAddressLoop@0	8	3.54	44.2%
+004200F5				_SearchEndMarkLoop@0	25	11.76	47.0%
-0042010E				_SearchSoundShaderCodeLoop@0	83	36.36	43.8%
0042010E				_SearchSoundShaderCodeLoop@0	83	36.36	43.8%
0042010E	AC	LODSB					
0042010F	FEC8	DEC	AL				
00420111	79FB	JNS	_SearchSoundShaderCodeLoop@0				
00420113	56	PUSH	ESI				
00420114	54	PUSH	ESP				
00420115	6A01	PUSH	0x1				
00420117	68B9910000	PUSH	0x91B9				
0042011C	FF154C004300	CALL	[_glCreateShaderProgramv@12]				
00420122	50	PUSH	EAX				
00420123	FF1550004300	CALL	[_glUseProgram@4]				
00420129	54	PUSH	ESP				
0042012A	6A01	PUSH	0x1				
0042012C	FF1554004300	CALL	[_glGenBuffers@8]				
00420132	BED2900000	MOV	ESI, 0x90D2				
00420137	6A00	PUSH	0x0				
00420139	56	PUSH	ESI				
0042013A	FF1558004300	CALL	[_glBindBufferBase@12]				
00420140	68EA880000	PUSH	0x88EA				
00420145	6A00	PUSH	0x0				
00420147	6800000004	PUSH	0x4000000				
0042014C	56	PUSH	ESI				
0042014D	FF155C004300	CALL	[_glBufferData@16]				
00420153	57	PUSH	EDI				
00420154	6800000004	PUSH	0x4000000				
00420159	6A00	PUSH	0x0				
0042015B	56	PUSH	ESI				
0042015C	BE00E07F00	MOV	ESI, 0x7FE000				

Crinkler compression report

Report for file C:¥usr¥dev_area¥demoscene¥4k_demo¥mini4k¥AsmAsmRelease¥mini4k.exe generated by Crinkler 2.2 on Fri Apr 10 22:43:09 2020

Options: /SUBSYSTEM:WINDOWS /ENTRY:entrypoint /COMPMODE:SLOW /HASHSIZE:300 /HASHTRIES:300 /ORDERTRIES:300 /UNSAFEIMPORT /OVERRIDEALIGNMENTS /UNALIGNCODE /NOINITIALIZERS

Output file size: 964

Bits per byte:

collapsed sections globals sections + globals globals expanded sections + globals expanded

Address				Label name	Size	Comp. size	Ratio
-00420000				Code sections	523	275.18	52.6%
+00420000					122	104.15	85.4%
+0042007A				_entrypoint@0	115	47.19	41.0%
+004200ED				_GetProcAddressLoop@0	8	3.54	44.2%
+004200F5				_SearchEndMarkLoop@0	25	11.76	47.0%
-0042010E				_SearchSoundShaderCodeLoop@0	83	36.36	43.8%
0042010E				_SearchSoundShaderCodeLoop@0	83	36.36	43.8%
0042010E	AC	LODSB					
0042010F	FEC8	DEC	AL				
00420111	79FB	JNS	_SearchSoundShaderCodeLoop@0				
00420113	56	PUSH	ESI				
00420114	54	PUSH	ESP				
00420115	6A01	PUSH	0x1				
00420117	68B9910000	PUSH	0x91B9				
0042011C	FF154C004300	CALL	[_glCreateShaderProgramv@12]				
00420122	50	PUSH	EAX				
00420123	FF1550004300	CALL	[_glUseProgram@4]				
00420129	54	PUSH	ESP				
0042012A	6A01	PUSH	0x1				
0042012C	FF1554004300	CALL	[_glGenBuffers@8]				
00420132	BED2900000	MOV	ESI, 0x90D2				
00420137	6A00	PUSH	0x0				
00420139	56	PUSH	ESI				
0042013A	FF1558004300	CALL	[_glBindBufferBase@12]				
00420140	68EA880000	PUSH	0x88EA				
00420145	6A00	PUSH	0x0				
00420147	6800000004	PUSH	0x4000000				
0042014C	56	PUSH	ESI				
0042014D	FF155C004300	CALL	[_glBufferData@16]				
00420153	57	PUSH	EDI				
00420154	6800000004	PUSH	0x4000000				
00420159	6A00	PUSH	0x0				
0042015B	56	PUSH	ESI				
0042015C	BE00E07F00	MOV	ESI, 0x7FE000				

この枠内が
ユーザーの
プログラムコード

Crinkler compression report

Report for file C:¥usr¥dev_area¥demoscene¥4k_demo¥mini4k¥AsmAsmRelease¥mini4k.exe generated by Crinkler 2.2 on Fri Apr 10 22:43:09 2020

Options: /SUBSYSTEM:WINDOWS /ENTRY:entrypoint /COMPMODE:SLOW /HASHSIZE:300 /HASHTRIES:300 /ORDERTRIES:300 /UNSAFEIMPORT /OVERRIDEALIGNMENTS /UNALIGNCODE /NOINITIALIZERS

Output file size: 964

Bits per byte: 0.1 0.5 1 2 3 5 7 9 12

collapsed sections globals sections + globals globals expanded sections + globals expanded

Address	Label name	Size	Comp. size	Ratio
-00420000	Code sections	523	275.18	52.6%
+00420000	_Import	122	104.15	85.4%
+0042007A	_entrypoint@0	115	47.19	41.0%
+004200ED	_GetProcAddressLoop@0	8	3.54	44.2%
+004200F5	_SearchEndMarkLoop@0	25	11.76	47.0%
-0042010E	_SearchSoundShaderCodeLoop@0	83	36.36	43.8%
0042010F	_SearchSoundShaderCodeLoop@0	83	36.36	43.8%

```
0042010E  AC            LODSB
0042010F  FEC8          DEC     AL
00420111  79FB          JNS     _SearchSoundShaderCodeLoop@0
00420113  56            PUSH    ESI
00420114  54            
00420115  6A01          
00420117  68B9910000    
0042011C  FF154C004300  derProgramv@12]
00420122  50            
00420123  FF1550004300  n@4]
00420129  54            
0042012A  6A01          
0042012C  FF1554004300  s@8]
00420132  BED2900000    MOV     ESI, 0x90D2
00420137  6A00          PUSH    0x0
00420139  56            PUSH    ESI
0042013A  FF1558004300  CALL    [_glBindBufferBase@12]
00420140  68EA880000    PUSH    0x88EA
00420145  6A00          PUSH    0x0
00420147  6800000004    PUSH    0x4000000
0042014C  56            PUSH    ESI
0042014D  FF155C004300  CALL    [_glBufferData@16]
00420153  57            PUSH    EDI
00420154  6800000004    PUSH    0x4000000
00420159  6A00          PUSH    0x0
0042015B  56            PUSH    ESI
0042015C  BE00E07F00    MOV     ESI, 0x7FE000
```

ネイティブコード
赤は高コスト
緑は低コスト

各セクションの
占有サイズと
圧縮率

シェーダコード
minify TIPS

シェーダコードの minify

- minify ツールは万能ではない
  - 最高の効率を求めるなら、手動 minify は避けられない。
- つぶやきGLSLに学ぶ
  - シェーダコードの minify については、https://twigl.app/ の貢献により開拓されまくった。
  - https://twitter.com/hashtag/つぶやきGLSL
- ただし、crinkler で圧縮する場合は、短くかつ素直なコードが有利
  - つぶやきGLSLのようなトリッキーな記述は、逆に規則性の無いコードとみなされ、圧縮効率ではマイナスに作用することがあるので注意。
  - 例えば、セミコロンをカンマで置き換えるようなテクニックも、統一性が下がり、意図に反して圧縮率が低下する場合がある。

関数オーバーロードを利用

- 関数オーバーロード
  - 引数構成や戻り値が異なる同一名の関数を複数定義可能な仕組み。
  - 本来は、引数型違いだが意味が同じ関数を同名で複数定義することで、コード可読性を上げるしくみ。
- 関数オーバーロードは minify で有効
  - 複数の関数名を同一名にでき、圧縮効率が向上する。
  - 全く用途の異なる関数に対して、同一名を割り当てることになるので、可読性は著しく低下。
- 関数オーバーロード化できる条件が難しい
  - 単に引数構成が異なれば良いというものではない
  - 衝突なくオーバーロード化できる条件は GPU ベンダごとに異なり互換性問題がある
  - shader minifier は互換性問題をクリアしつつ自動でオーバーロード化してくれる

定義済みシンボルを再定義

- スコープが異なるなら同一名のシンボルが再定義できる
  - 例

```
// 二つの a は異なる変数とみなされる。
float a;
{
        float a;
}
```

- 通常は、コード視認性のため、それぞれ異なる変数名を割り当てるべきだが、同一変数名を使うことで圧縮率を高めることができる。

- shader minifier はこのような最適化を行わないので、自力でやるしかない。

変数名は使用傾向を統一

- 用途ごとに変数名を固定することで、似た傾向のプログラムコードを生成し、圧縮率を高めることができる。
  - 例
    - for ループのカウンタは必ず i を使う
    - 座標を意味する変数名は必ず p を使う
- shader minifier はこのような最適化を行わないので、自力でやるしかない。

int 型を float 型に置き換える

- float 型に置き換え可能な int 型は、float 型に統一した方が有利な場合が多い。
- “float” は “int” よりも 2 文字長いが、より小さい圧縮結果が得られる傾向がある。
  - float は頻出キーワードなので極めて低コスト
  - int は頻出キーワードでないので高コストになりやすい
- for 構文のループ変数は float にしてみることを検討する

同一の記述が連続で出現しやすい工夫

- 例
  - min() max() 関数の引数は、順序を入れ替えても結果は同じ。
  - 引数の順序を入れ替えることで、圧縮率を高めることができる場合がある。
    - 例

      修正前：min(...,min(...,min(...,...)))

      ↓

      修正後：min(min(min(...,...),...),...)

      min を連続させることで、圧縮率が向上する。

ネイティブコード minify TIPS

実行ファイル全体をアセンブラで記述

- C/C++言語はコードサイズ最小化を狙う上ではかなり不利
- 実行ファイル全体をアセンブラで記述しなければならない
- 短い命令で書きたい都合上、X86 命令を使う

なるべく短い命令を使う

- X86 は命令の種類によって占有バイト数が異なる。

- なるべく短いバイト数で表現できる命令を使う。
  - 例
    - レジスタの 0 クリアは XOR 命令で可能

      ```
      XOR         EAX, EAX            ; 2バイト命令
      ```

    - レジスタ値代入だと 2 バイト命令だが、レジスタ値交換だと 1 バイト命令で済む。

      ```
      XCHG        ECX, EAX            ; 1バイト命令
      ```

    - 等々、書ききれないぐらい様々な方法が存在

単純な命令の組み合わせに置き換え

- 例

  EAX レジスタに ESI レジスタを代入したい場合、ナイーブな実装だと

  ```
  MOV     EAX, ESI    : 2バイト命令
  ```

  となる。これは、

  ```
  PUSH    ESI         : 1バイト命令
  POP     EAX         : 1バイト命令
  ```

  に置き換えたほうが有利な場合が多い。

  どちらの書き方も 2 バイトになるが、後者の方が使用頻度の高い単純な命令の組み合わせで表現されるため、圧縮率を高めることができる。

レジスタは使用傾向を統一

- 例えばスタックに何か値を push することを目的にレジスタに値を保持する場合、必ず特定のレジスタを使う、というような工夫を行う。
- レジスタ使用傾向を統一することで、似たプログラムコードが生成され、圧縮率を高めることができる。
- シェーダコードの minify において変数名を統一するのと同じ理屈。

entrypoint のレジスタ初期値を知る

- crinkler が生成した _Import 処理のあと、ユーザープログラムの entrypoint が実行される。

- entrypoint 突入時点のレジスタの状態を調べ、利用可能なものは利用する。

  EAX = 不定

  EBX = 不定

  ECX = 0 (loop 命令を抜けた後なので)

  EDX = 不定

  ESI = _DLLNames から始まる dll 名テーブル末端の 0xFF の次

  EDI = _ImportList から続く関数アドレステーブルの次

  EBP = 0（LoadLibrary() の戻り値 false が格納されている)

entrypoint のレジスタ初期値を活用

- ECX & EBP レジスタ
  - 0 が入っている。とりあえず 0 が欲しいケースで使える。
- EDI レジスタ
  - BSS セクションの先頭を指している。
- ESI レジスタ
  - crinkler の DATA セクションの次を指している。
  - crinkler の DATA セクションの直後にユーザーの DATA セクションを配置できれば有用。
    - 残念ながら、必ずこの配置に確実に固定する妥当な方法がない。
- crinker は恐らくこのようなユースケースを想定してレジスタ初期値を用意してくれている。

自力でAPIアドレス取得する方が有利な例

- crinkler は、API 名を 4 バイトハッシュで表現している。
  - したがって、API 名解決 1 件につき、4 バイト消費する。
- ユーザープログラム側で OpenGL を利用する場合、大抵 wglGetProcAddress() を使って GL の拡張 API アドレスを取得している。
- つまり、2種類のAPI アドレス取得方法が混在している状態になる。
  - どっちが有利か？比較する必要がある。
- glRects のような短い名前の API の場合、crinkler に任せるより wglGetProcAddress() で取得した方が、サイズで有利な場合がある。
  - 注）環境によっては GL 拡張命令以外の API アドレスが取得できずうまく行かない

Win32 API 等の戻り値を活用

- API が成功したのか失敗したのかを示す戻り値を別の意味に利用。
- 例えば EAX レジスタに 0 を返す API は、EAX レジスタのゼロクリアの代わりに使える。
- 意外に貴重なのが EAX レジスタに 1 (true) を返す API。
  - レジスタに 1 を格納するのは最低でも 3 バイトのコードが必要で高コスト。
  - 戻り値の 1 が流用できるならラッキー。

pre push

- 関数呼び出し時の引数 push は、呼び出し直前である必要は必ずしもなく、前もって実行しておくことが可能
  - 複数の関数の引数をまとめてあらかじめ push しておいても良い。
- これを pre push と呼ぶ。
- 似た引数は、まとめて積むことで、似た命令を連続させることができ、圧縮率を高めることができる。
  - Win32 API は、やたら引数に 0 を要求するケースが多く、それらをまとめることで高い圧縮効果が得られる。

アンドキュメンテッドな API 仕様の活用

- Win32 API には、ドキュメントに記述がないが、過去バージョンとの互換のため残されている謎仕様が存在する
  - これらを活用することで minify できる場合がある

- 例：CreateWindowEx() の引数
  - 第二引数 lpClassName には、本来は “edit” などの文字列のポインタを指定する。
  - この第二引数は謎仕様があり、ポインタではなく、0xC018 などの ATOM と呼ばれる謎値を指定しても、“edit” を指定したものと同じ効果がある（危険な香りがするが）。
  - このような謎値は Microsoft のドキュメントに記述がないが、重要 minify テクニックとなっている。

省略して良い API を探す

- システムのデフォルトステートが、ユーザープログラム側で希望するステートそのものであることが明らかなら、ステート設定 API を省略できる。
  - 例
    - OpenGL では、テクスチャ 0 番はデフォルトで存在し、バインドもされている。

- 内部で大した処理を行っていない API は自前処理で代替する
  - 例
    - waveOutPrepareHeader(hWaveOut, lpWaveHdr, sizeof(WAVEHDR));
    - この API 内部でやっていることは、メンバ dwFlags に WHDR_PREPARED を設定することだけ。
    - 自力でフラグだけ立てて、API 呼出しを省略しても大丈夫。
    - ドライバのベンダ実装次第では省略できない可能性はあるが、省略により副作用が起きたという例は知りうる限りでは存在しない

crinkler の _Import に含まれる処理の流用

- _Import とは（おさらい）
  - crinkler で圧縮した exe には必ず含まれる、dll に含まれる API アドレスを取得する処理。

- _Import に含まれるバイト列と同じコードを使うと、高い圧縮率が期待できる。
  - 例
    - _Import は以下のような処理を含む。
      - do { byte = *p; p++; --byte; } while (byte >= 0); ++byte;
      - 文字列中に負の値が出現するまでポインタを移動するという単純な処理である。
    - 似た処理は、ユーザープログラム側でも出現しやすい
      - OpenGL の API 名リストからアドレス取得など
    - 使用レジスタやデータ構造を一致させておくことで、crinkler _Import の処理と全くおなじバイト列のコードを出現させ、高い圧縮率を得ることができる。

小さな配列変数はスタック上に作る

- 配列変数が必要なコードをナイーブに書くと、DATA セクション上に配列実体を作ることになる。
  - 配列実体のアドレスを取得する必要が生じ、コードサイズが肥大する。

- 小さな配列は、スタック上に作ってしまった方がコードサイズ面では有利。
  - 例

    配列変数

    ```
    int args[] = {1,2};
    ```

    をスタック上に作成するコード。わずか 4 バイトで書ける。

    ```
    PUSH  2          ; 2バイト命令
    PUSH  1          ; 2バイト命令
    ```

    ESP レジスタが、 {1,2} を格納した配列の先頭アドレスになっており、アドレス取得のコストもゼロ。

使用済みデータは破壊して良い

- 今後使用される見込みのないデータは破壊しても良い。
  - 例
    - コンパイル済みシェーダコードなどが該当。
    - 破壊しても副作用がない。
    - シェーダコンパイル終了後、シェーダコード先頭を指すレジスタは、広大なワークメモリ先頭アドレスを保持するとみなしてよい。

ストリング操作命令の活用

- EDI ESI レジスタは、元々文字列の加工に特化した専用命令が利用可能。
  - 例
    - STOSD（1 バイト命令）

      MOV [EDI], EAX

      ADD EDI, 4

      と互換
    - LODSB （1 バイト命令）

      MOV AL, [ESI]

      ADD ESI, 1

      と互換

- ポインタ経由で読み書きする多くのケースで有用。

文字列サイズは実行時評価した方が有利

- 通常のプログラミングでは、各文字列の先頭アドレスやサイズなどは、コンパイル時点で確定したくなるだろう。
- しかし実際には、それぞれの文字列について先頭アドレスまたはサイズをコード中に保持するのはコストが高い。
- 文字列はあらかじめ必要順に連結しておき、その都度シークした方が有利な場合がある。
  - 文字列終点のシーク処理は crinkler の _Import にも存在するため圧縮が効きやすい。

API に渡す構造体サイズを詐称

- Win32 API の中には、API の世代を判定するため引数として構造体サイズを要求するものがある。
  - 引数として渡した構造体サイズは API のバージョン判定として利用される。
  - しかしレガシーな Win32 API が今後拡張されることはないため、この仕組みは形骸化している。

- つまり、構造体サイズは規定値以上ならでたらめな値を指定可能
  - 例
    - waveOutGetPosition(hWaveOut, &mmTime, sizeof(MMTIME));
    - sizeof(MMTIME) は 12 だが、12 以上の値を指定しても動作するし、副作用はない。
    - 周辺で 12 以上の値を push しているコードを探し、12 の代わりにそれと同じ値を push しておくことで、似たコードを出現させ、わずかに圧縮率を改善できる。
    - 12 以上であることが分かっているなら、何かしらのレジスタ上に載っているゴミを push しても良い。

構造体の共有

- レガシーな Win32 API が引数として要求する構造体には、ユースケースによっては未使用となるメンバが沢山ある。

- 未使用メンバが存在する領域に、別の構造体を埋め込むことが可能。

ESCキーが押されたら終了させる処理

- GetAsyncKeyState() の戻り値は 16 bit であり、キーが押されると最上位ビットと最下位ビットが 1 になる（最下位ビットは、前回呼び出し時から新規にキーが押されたときのみ）

- EAX = GetAsyncKeyState() の戻り値 だとして、ESC が押されているならループを抜ける処理を考える。
  - ナイーブな実装だと、

    ```
    TEST   AL, AL       ; AL レジスタが 0 か確認 （2バイト命令）
    JZ     Loop         ; 0 なら Loop（2バイト命令）
    ```

  - フラグレジスタを利用すると1バイトの節約になる

    ```
    SAHF                ; AH -> flags（1バイト命令）
    JNS    Loop         ; sign フラグが 0 なら Loop （2バイト命令）
    ```

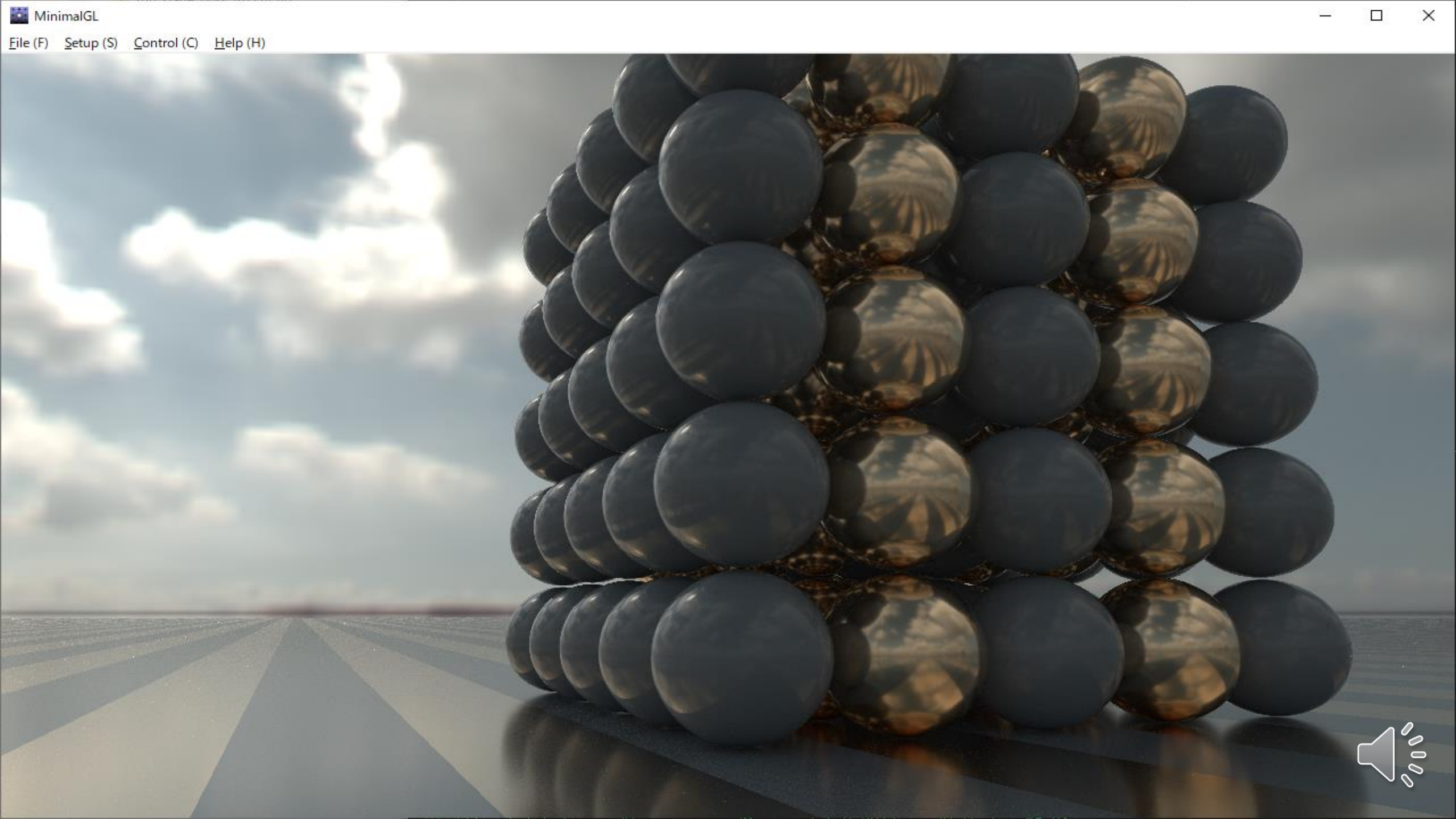
IEEE754 浮動小数の下位ビットの活用

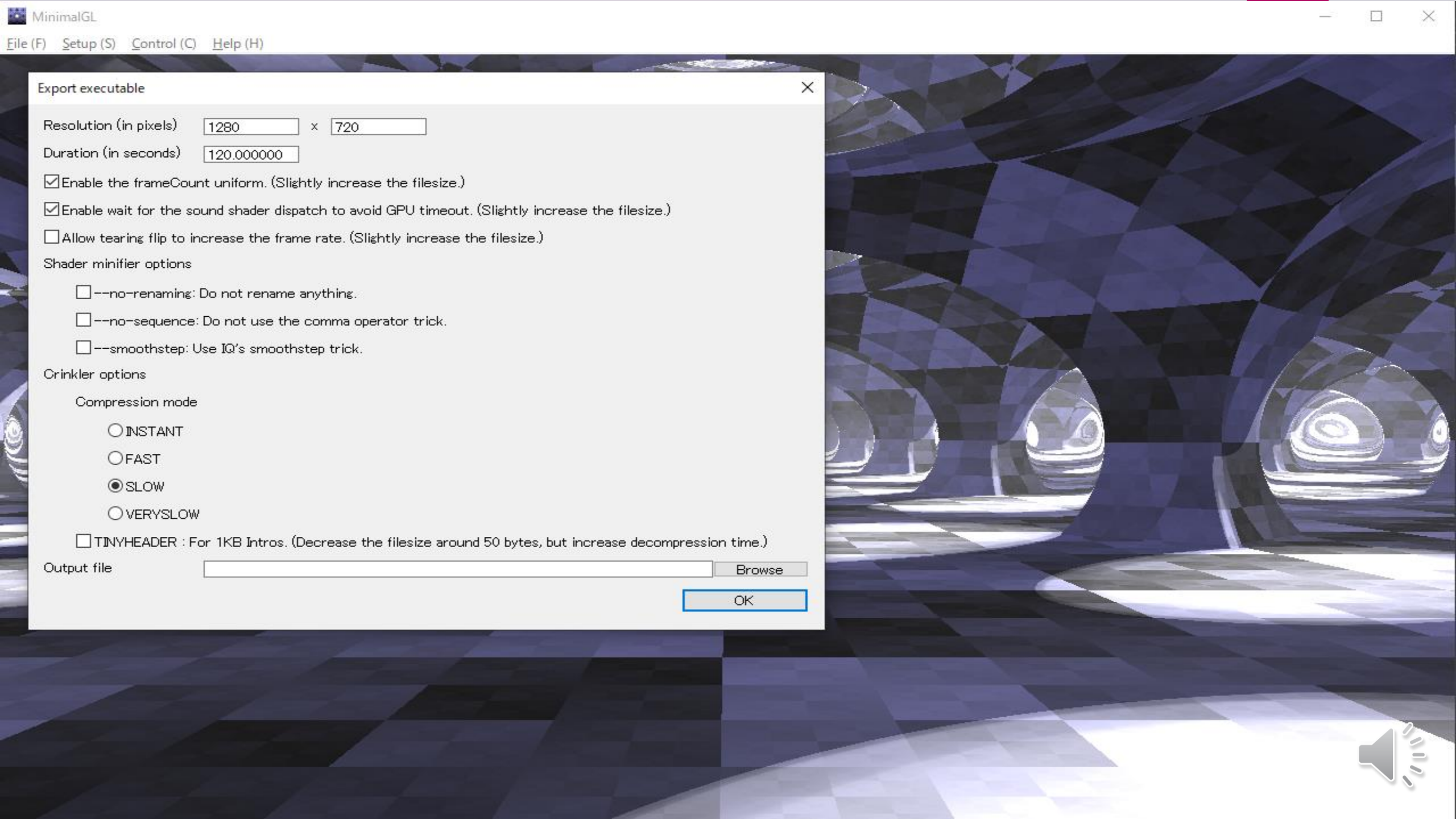
- float の下位ビットは多少のゴミが乗っていても、誤差が生じるだけで、致命的な問題にはならないケースがある。
- 下位ビットは別の用途に転用できる。
- 2 つの float 配列を開始アドレスを 2 バイトずらし重ねてメモリ配置することも可。
  - 仮数部 7 ビット精度になるが、ユースケース次第では問題ない。
- float の下位ビットをゼロクリアし、精度を犠牲にして圧縮効率を上げることも可。
  - crinkler に /TRUNCATEFLOATS を指定すると、これを自動で行ってくれる。

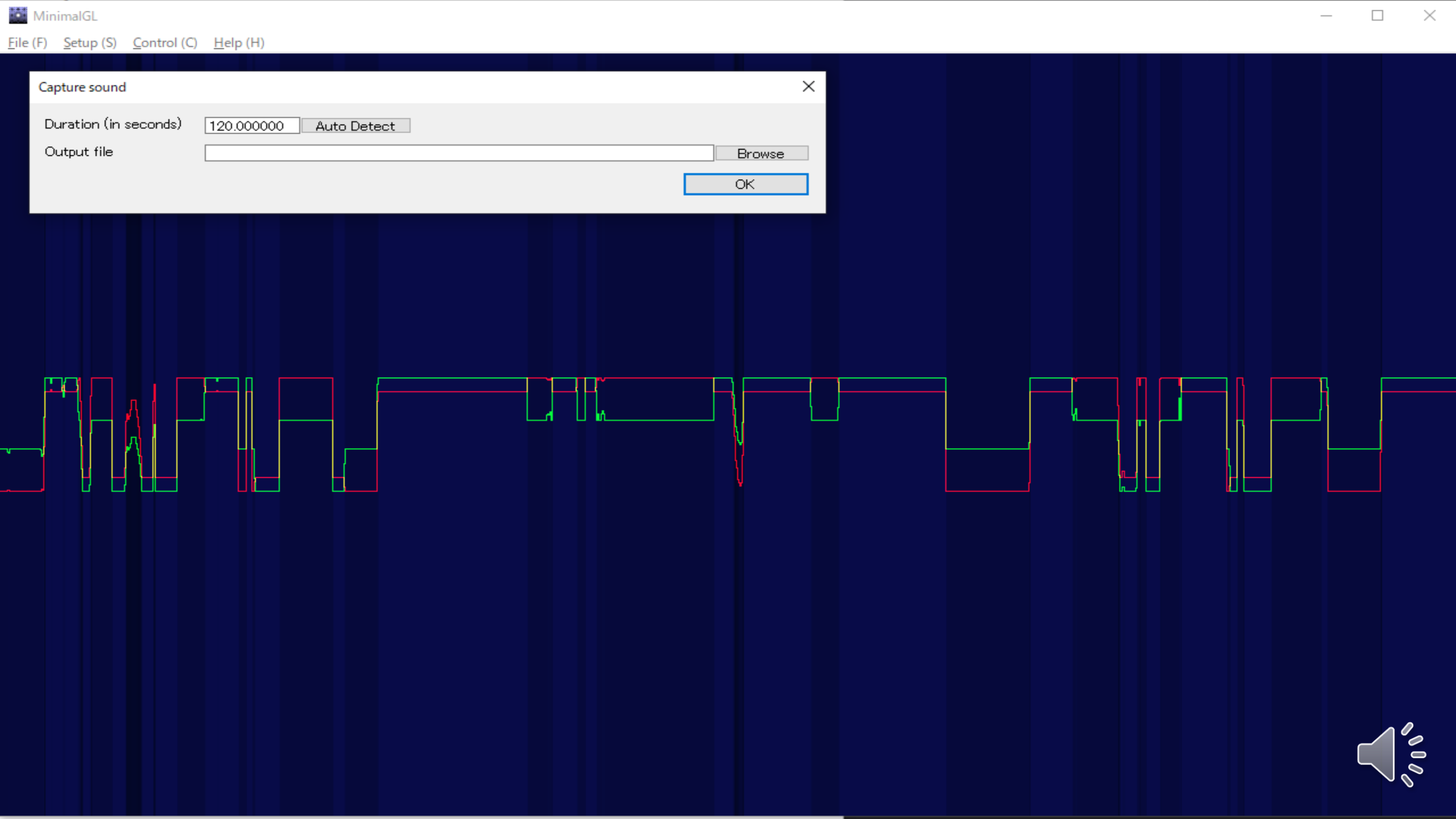
MinimalGL

MinimalGL

- 拙作の PC 4K Intro エディタ。
  - シェーダホットリロード可能な簡単なビューワ＋exe エクスポーター
  - https://github.com/yosshin4004/minimal_gl
- ここまでで解説したネイティブコードの minify テクニックの多くは、MinimalGL に組み込まれています。
  - ただし、アセンブラを使った minify は、exe エクスポート設定などのユースケース毎にハードコードが必要なため、入っていません。
- ぜひご活用ください。


MinimalGL
File (F)
Setup (S)
Control (C)
Help (H)


MinimalGL
File (F)
Setup (S)
Control (C)
Help (H)
Export executable
Resolution (in pixels)
1280
x
720
Duration (in seconds)
120.000000
Enable the frameCount uniform. (Slightly increase the filesize.)
Enable wait for the sound shader dispatch to avoid GPU timeout. (Slightly increase the filesize.)
Allow tearing flip to increase the frame rate. (Slightly increase the filesize.)
Shader minifier options
--no-renaming: Do not rename anything.
--no-sequence: Do not use the comma operator trick.
--smoothstep: Use IQ's smoothstep trick.
Crinkler options
Compression mode
INSTANT
FAST
SLOW
VERYSLOW
TINYHEADER : For 1KB Intros. (Decrease the filesize around 50 bytes, but increase decompression time.)
Output file
Browse
OK


MinimalGL
File (F)
Setup (S)
Control (C)
Help (H)
Capture sound
Duration (in seconds)
120.000000
Auto Detect
Output file
Browse
OK

謝辞

本資料は、先駆者の方々の成果・知見の上に成り立っています

- Demo-Framework-4k by Inigo Quilez
  - https://madethisthing.com/iq/Demo-Framework-4k
- Crinkler by Mentor/TBC and Blueberry/Loonies.
  - https://github.com/runestubbe/Crinkler
- Shader minifier by LLB / Ctrl-Alt-Test
  - https://github.com/laurentlb/Shader_Minifier
- pouet の size coding articles
  - https://www.pouet.net/
- その他多くの PC Intro 作品群

おわり