令和98年度 プロジェクトデザインIII

WebAssemblyを用いたグラフィクスレンダリングの高速化

4EP4-04

天羽大樹（あもうたいき）

令和99年99月99日

KIT 金沢工業大学

1. はじめに – 背景と目的 –

- 将来、Webアプリケーションとして、リアルタイムに変化するxRコンテンツを作成し、動作させる際、グラフィクスレンダリングの速度が遅くなってしまうと、没入感を損なってしまう問題があると考えられる
- そのような問題に対処するために、Web上でグラフィクスのレンダリングを早く処理するために、どのような手法を用いれば、より早く処理できるのかという取り組みを行う必要がある
- 本プロジェクトにおいては、WebAssemblyという仕組みを用いたバイナリフォーマットプログラムを使って、グラフィクスレンダリングを早く処理させるプログラムを作成する

発表の流れ

1. はじめに – 背景と目的 –
2. 用語などの簡潔な説明
3. プログラム概要
4. 性能比較評価・考察
5. むすび

2. 用語の簡潔な説明

- WebAssembly
  Web上で動作するバイナリフォーマットプログラムのこと。
  wasmと略して呼称されることが多いため、本スライドでも
  その略称を以後使用する。特定のプログラミング言語で記述した
  プログラムをそのバイナリフォーマットにコンパイルしたものを
  主に指す。そのプログラムを保存したファイルは、
  wasmモジュールと呼称されることが多い。

- WebGL
  Web上でグラフィクスを扱うために、ブラウザに組み込まれているAPI
  後述するWebGPUより歴史が古く関連ライブラリが豊富で、
  Three.jsやBabylon.jsといったライブラリが有名。
- WebGPU
  WebGLより高度にGPUの性能を活かし、高速なグラフィクス
  レンダリングを行うことが可能であるとされている
  ブラウザ組み込みのAPI。まだ公開されてから日が浅いため、
  著名な関連ライブラリなどはまだあまり見受けられない

発表の流れ

1. はじめに – 背景と目的 –
2. 用語などの簡潔な説明
3. プログラム概要
4. 性能比較評価・考察
5. むすび

3. プログラム概要

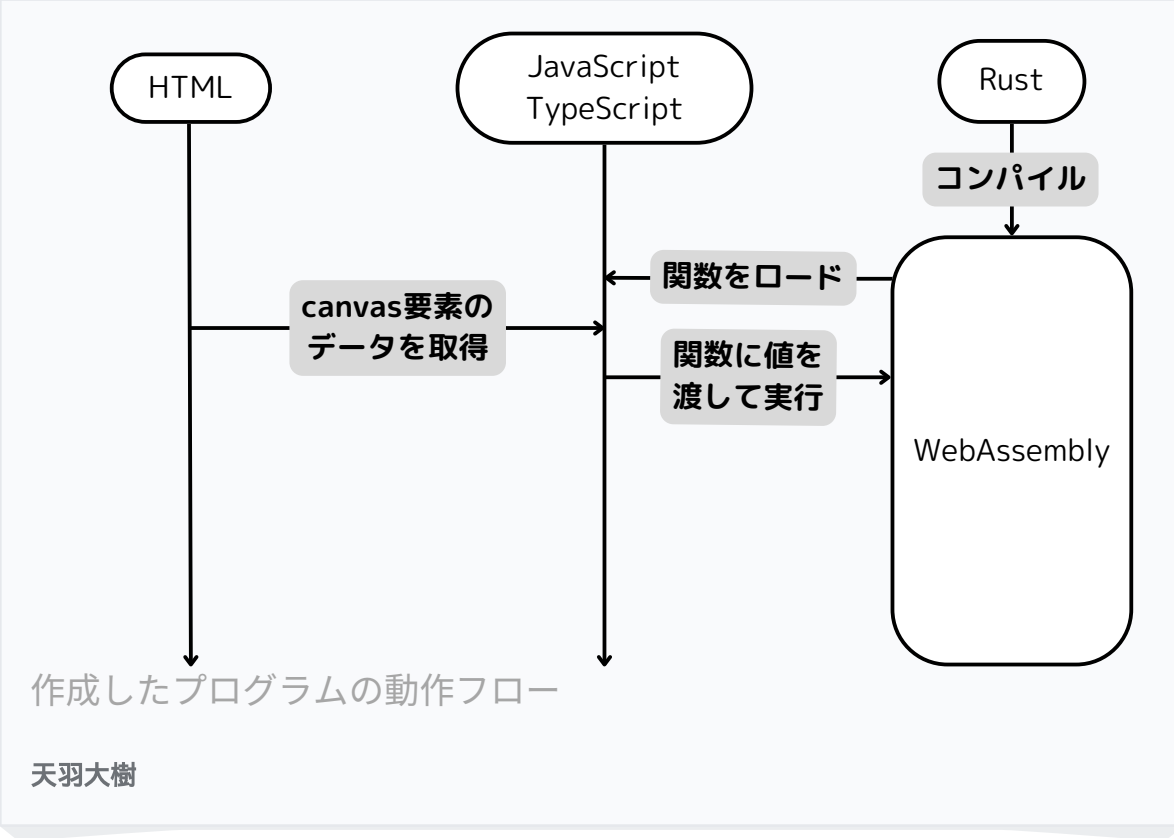
Rustでブラウザに搭載されているAPIにアクセスするプログラムを記述し、
そのプログラムをWebAssemblyにコンパイルする。
コンパイルされたプログラムをJavaScriptでロードし、実行させる。
WebAssembly側で処理するプログラムの対象として、

- 比較的、ハードウェア側に近いプログラムの処理
- 行列などを扱う演算負荷の高い処理

というような処理を指定して行う

本プロジェクトでは、2つのプログラミング言語を用いて高速化を行う

1. Rustのプログラムでグラフィクスを操作するAPIをコール
2. RustのプログラムをWebAssemblyにコンパイルする
3. WebAssemblyプログラムをJavaScript側からコールする



作成したプログラムの動作フロー

天羽大樹

3-1. 作成したWebAssemblyプログラムの概要

処理内容

WebGPU APIを通してレンダリング内容を決め、実行する内容をバッファデータとして処理し、順番に実行させる一連の処理を行う

実行される処理の結果

HTMLのcanvas要素の範囲にプログラムで指定した
矩形や図形がレンダリングされる

発表の流れ

1. はじめに – 背景と目的 –
2. 用語などの簡潔な説明
3. プログラム概要
4. 性能比較評価・考察
5. むすび

4. 性能比較評価・考察

評価方法

JavaScriptのみで組んだ関数と、WebAssemblyを交えたプログラムを同時に実行して、その2つの関数の実行が終了するまでの時間を計測し、比較を行う

評価結果

今回の比較では、以下の結果が出た。このケースではそれほど大きな差が出なかった。

考察

今回は単純な単色の図形、もしくは三色のグラデーションの図形のみの描画であるため、大きな差にはならなかったと考えられる。しかし、2つの比較で、それぞれ速度差が現れているため、より内部の数値処理を増やすと、より大きな差が生まれる可能性があると考えられる

5. むすび

- Web上でのグラフィクスレンダリングを高速化するため、WebAssemblyを用いたモジュールを作成した。
- 現時点では、それほど大きな描画速度の差がないが、計算する数値が増えれば増えるほど、差が開いていく可能性があると考えられる。
- 来月の報告までに、より計算量が多くなるプログラムで比較を行う

ここからおまけ

PDFファイルと同じフォルダにdemo002.mp4があれば再生できる.

YOUTUBEで再生

https://youtu.be/74agBeJxdFI

リスト 1: test2.c

```
1  #include <avr/io.h>
2  #include <avr/wdt.h>
3  int main(void)
4  {
5     DDRC = 0x30; // PC5/4を出力ピンに設定
6     PORTC = 0x10; // PC5/4の出力をL/Hに設定
7     for (;;) {
8        wdt_reset(); // ウォッチドックタイマをリセット
9     }
```

```
10      return 0;
11  }
```

リスト 2: test2.py

```
1  from time import sleep
2  from random import randint
3
4  while True:
5      input('push ENTER key')
6      r = randint(1,6)
7      print( r )
8      sleep(0.5)
```

UNIXv1におけるタスク切り替えが行われるタイミング

① みなさん

② こんにちは

- まんじゅう
- りんご

③ お元気で
またあうひまで

```
$ gcc test.c ↵
 (*_*)
 (*_*)
```

ここでCTL+Cを押す

謝辞 本研究はJSPS科研費21Kxxxxxxxxx助成を受けた

文献

[1] K.Thompson，D.M.Ritchie，**”The UNIX Time-Sharing System”**，Communications of the ACM, Vol.17, No.7, 1974.

[2] Digital Equipment Corporation: **PDP11/20-15-r20 Processor Handbook**, 1971.

[3] T.R. Bashkow, **”Study of UNIX: Preliminary Release of Unix Implementation Document”**, `http://minnie.tuhs.org/Archive/Distributions/Research/Dennis_v1/PreliminaryUnixImplementationDocument_Jun72.pdf`, Jun. 1972.

[4] simh, **”The Computer History Simulation Project”**, `https://github.com/simh/simh`, 参照 Mar.14, 2022.

[5] W.Toomey, **”First Edition Unix: Its Creation and Restoration”**, IEEE Annals of the History of Computing, 32 (3), pp.74-82, 2010.

[6] Diomidis.Spinellis，**”unix-history-repo”**, `https://github.com/dspinellis/unix-history-repo/tree/Research-V1`, 参照 Mar.14, 2022.

[7] Digital Equipment Copporation: **PDP11 Peripherals HandBook**, 1972.