令和6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6年09月25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ブラウザ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ブラウザに搭載されている仮想マシンで
  実行できるバイナリコードとそれを処理するシステム全体[1]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コード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。
  C,C++,Rustなどで記述したプログラムをこのWebAssemblyに
  コンパイルすることで、動作させることが可能になる。[2]

- 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

2-1. なぜWebAseemblyを扱うのか

扱うメリット

WebAssemblyはWebブラウザ上でネイティブコードに近い速度で実行される[3]ため、JavaScriptでは時間がかかるであろう処理をWebAssemblyでの処理に置き替えることによる高速化が望める

WebAssemblyが得意とする処理

基本的に、WebAssemblyそのものは、整数・実数の計算しかできない[4]が、WebAssemblyが既にコンパイルされた仮想マシンコードであることを活かし、高速な演算処理を得意とする。

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

3. プログラム概要

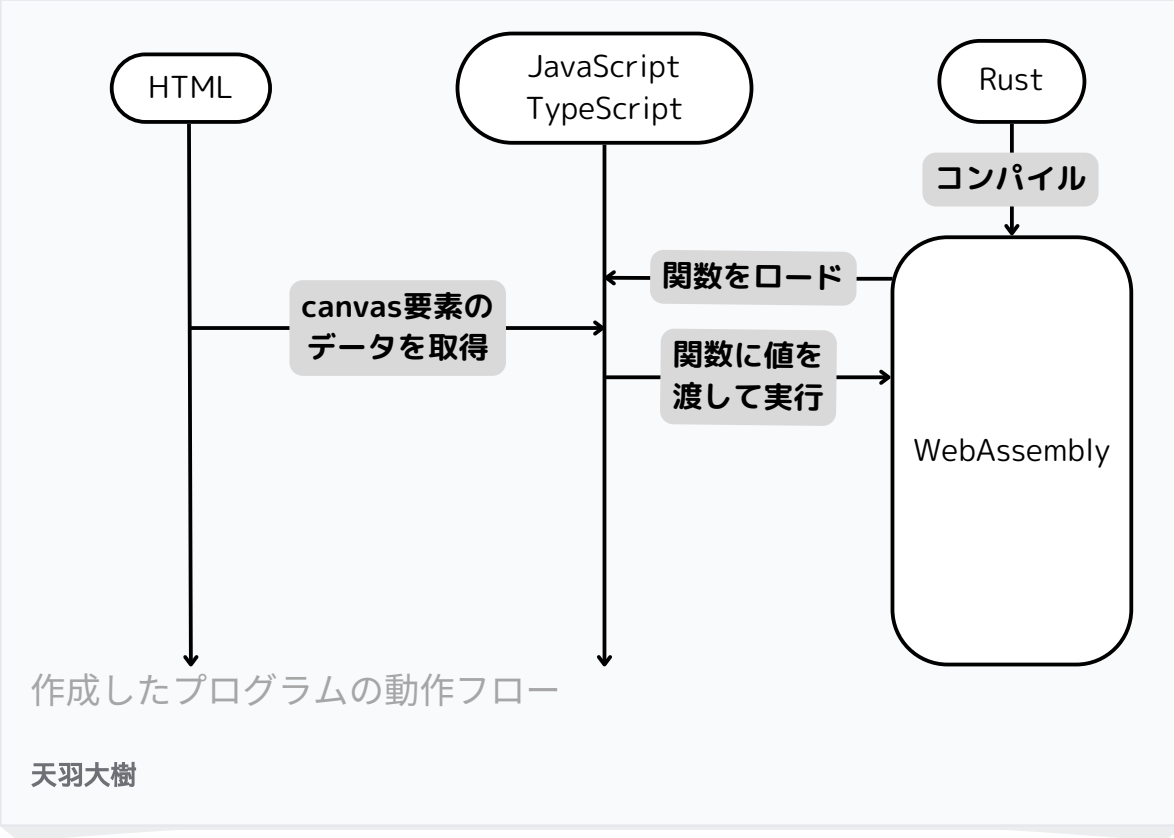
Rustでブラウザに搭載されているAPIにアクセスするプログラムを記述し、
そのプログラムをWebAssemblyにコンパイルする。
コンパイルされたプログラムをJavaScriptでロードし、実行させる。
WebAssembly側で処理するプログラムの対象として、

- 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

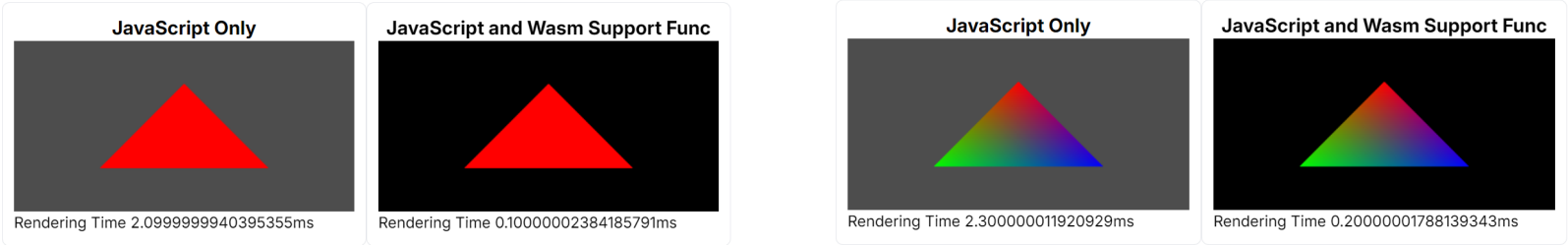
発表の流れ

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

評価方法とその手順

次ページの画像のような色の異なる三角形をそれぞれレンダリングする際、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その2つの関数の実行が終了するまでの時間をJavaScriptのperformanceクラスのメソッドを利用して、
実行を終了した時間から実行を開始した時間を減算することで計測。
これを、for文で500回行った実行時間データをCSVデータとし、Python言語を用いて平均値、中央値、最大値、最小値を算出する。また、Pythonライブラリである、**matplotlib.pyplot**を使用し、箱ひげ図を作成することにより、安定的に高速化が可能かどうかを評価する



比較対象に関する詳細な説明

比較対象は、描画プログラムを以下のように記述したものである。

- 左画像の左側が、JavaScriptのみで記述したもの
- 左画像の右側が、JavaScript+Wasmで記述したもの
- 右画像の左側が、JavaScriptのみでカラフルな図形をレンダリングするように記述したもの
- 右画像の右側が、JavaScrpt+Wasmでカラフルな図形をレンダリングするように記述したもの

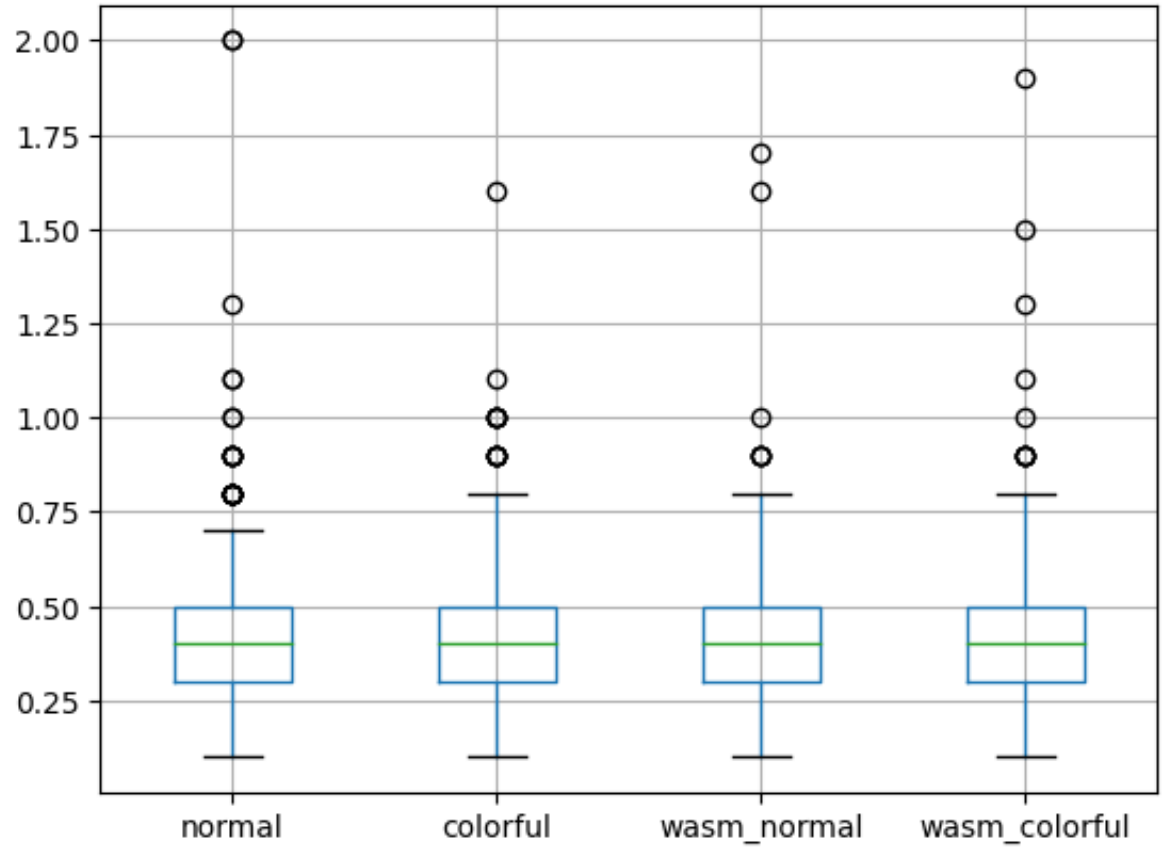
評価結果

以下の表には、計算した4つの値について記載する。
時間の単位については全てミリ秒である。

描画内容	Wasmの有無	平均時間	最小時間	最大時間	時間の中央値
赤色三角形	無	0.445691	0.1	2.0	0.4
赤色三角形	有	0.424649	0.1	1.6	0.4
虹色三角形	無	0.413427	0.1	1.7	0.4
虹色三角形	有	0.416232	0.1	1.9	0.4

右記の箱ひげグラフの結果から、以下のようなことが言える

1. 箱ひげ自体に着目すると、大きな変化があるようには見受けられない
2. 赤色の図形は最遅の場合の時間が短縮されているが、それ以外の目立った違いが見受けられない
3. 虹色のもの場合は、最遅のケースを比較した際、実行時間が増えてしまっている。



考察

赤色三角形の場合

平均値や最大値のデータやグラフからは、単純かつ単色の図形のレンダリングであれば、高速化が見込める可能性があるといえる

虹色三角形の場合

虹色三角形の場合は、平均値や最大値のデータやグラフから、
むしろ遅くなってしまっていることがわかる。

結果から

矩形自体のレンダリングに関しては高速化が可能であると考えられるが、色の複雑な処理が入ってしまうと、むしろ減速してしまうため、カラフルな描画対象を扱う状況にには不向きであると考えられる

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、1つの図形しか描画していないため、それほど大きな描画速度の差がないが、描画物を増やしたらどう変化するのかを検証しようと考えている。
- 来月の報告までに、図形の量を増やすことや、プログラム内容の質を高め、より負荷が高くなるプログラムで比較を行う

文献

[1] 日向俊二. (2021). より速く強力なWebアプリ実現のためのWebAssemblyガイドブック (初). カットシステム.

[2] [1]と同様

[3] Gallant, G. (2022). ハンズオン WebAssembly ——Emscripten と C++を使って学ぶ WebAssemblyアプリケーションの開発方法 (初). オライリー・ジャパン.

[4] [1]と同様