Vol.2016-MBL-80 No.12
Vol.2016-CDS-17 No.12
2016/8/25

空間補間を用いたストリーミングセンサデータ向けリアルタイム可視化システムの提案

若森和昌[†1]　丸島晃明[†2]　峰野博史[†3]

概要：無線センサネットワークの普及に伴い，様々なセンサデータが収集可能となっただけでなく，人間がセンサデータを視覚的に理解することを目的としたセンサデータ可視化システムの開発もさかんに行われている．中でも空間補間を用いた可視化は，WSN 設置空間の視覚的理解を更に促進する技術として注目されている．しかし，空間補間を用いた可視化の処理量は膨大であることから，既存システムでは，加速度データ等の高レートなストリーミングセンサデータを，リアルタイムに空間補間し可視化することは難しかった．本研究では，高レートに生成されるストリーミングセンサデータに対しても，空間補間を用いてリアルタイムに可視化できるシステムを提案する．本システムは，空間補間を並列処理し，WebGL を用いて描画することで，可視化に要する処理時間を短縮し，リアルタイムな可視化を実現する．プロトタイプとして 18 台の無線加速度センサノードを用いた振動可視化システムを開発し基礎評価を行ったところ，60 Hz で生成されるストリーミングセンサデータをフレームレート 60 fps でリアルタイムに可視化できることを確認した．また，吊橋における屋外評価において振動の伝播をリアルタイムに可視化でき，本システムを用いて建造物の振動をリアルタイムに分析できる見通しを得た．

1. はじめに

近年，センサ技術や無線ネットワーク技術の発展に伴い，無線センサネットワーク（WSN）の普及が進んでいる．特に，無線ネットワークの伝送速度が向上したことで，加速度といった高レートでサンプリングされるセンサデータをリアルタイムに収集可能となった．高レートで絶え間なく伝送され続けるセンサデータをストリーミングセンサデータと呼び，将来は IoT デバイスの普及やスマートシティが実現され，生成されるストリーミングセンサデータ量は増大することが予想される．そのため，増大するストリーミングセンサデータのリアルタイム活用技術の発展は重要な課題である．

センサデータの活用例として，センサデータ可視化システム[1-6]がある．可視化システムは，センサデータを人間が視覚的に理解可能な形式に変換し，ユーザへ提示するシステムである．その中でも，空間補間を用いた可視化はセンサを配置した空間の状態をヒートマップ状に描画するため，人間が空間の状態を直感的に理解可能となる[5]．空間補間を用いた可視化ではヒートマップの粒度を高精細にすることで滑らかな可視化となり，更なるユーザエクスペリエンスの向上が期待できる．しかし，高精細な可視化を行う場合，システムの可視化処理である空間補間と描画の処理量は増大する．また，ストリーミングセンサデータは高レートで生成されるため，ストリーミングセンサデータのリアルタイム可視化において許容できる可視化処理時間は限られている．そのため，空間補間を用いたストリーミングセンサデータの可視化において，リアルタイムかつ高精細な可視化を実現するには可視化処理時間の短縮が重要となる．

本研究では，空間補間を用いた可視化システムにおける可視化処理である空間補間と描画の処理時間の短縮に重点を置き，高レートで生成されるストリーミングセンサデータをリアルタイムに空間補間し，高精細に可視化できるシステムを提案する．本システムでは，WebWorkers[7]を用いて空間補間を並列処理し，WebGL[8]を用いて高速に描画することで，可視化処理を低遅延で実行可能とする．

以降，2 章で関連研究について述べ，3 章で空間補間を用いたストリーミングセンサデータ向けリアルタイム可視化システムの提案を行い，4 章でプロトタイプシステムの実装について述べる．5 章にて基礎評価結果，6 章にて屋外評価結果を述べ，最後に 7 章で本論文をまとめる．

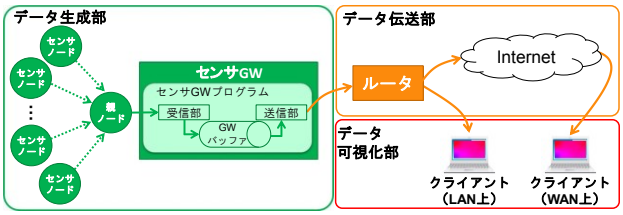
2. 関連研究

WSN を用いたセンサデータ可視化システムは多数提案されている[2-3]．消費電力や人間行動を可視化することですることで省電力化を促すシステム[2]や，屋外の温度等の環境データを可視化することで，一般市民に周囲の環境の理解を促進させ，緑地計画に利用するシステム[3]がある．これらのシステムは，可視化方法をグラフや計測点のマッピングとしており，空間補間を用いていない．特に環境データ可視化システム[3]では，空間補間を用いて可視化することで，市民の環境の理解を更に促進できると考える．

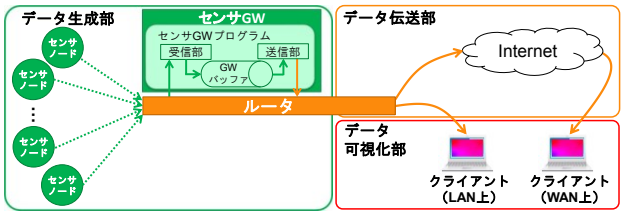
センサデータの空間補間を用いた可視化システムとして温湿度可視化システム[4-5]がある．温湿度可視化システムは WSN を用いて温湿度を収集し，クライアント端末にて空間の温湿度分布を可視化する．空調設備を設置した室内にてデータ収集を行い，時間の経過に伴う室内の温湿度変化をヒートマップ等で可視化することで，空調設備の性能や効果を手軽かつ適切に評価可能であることが示されて

†1 静岡大学情報学部
Faculty of Informatics, Shizuoka University
†2 静岡大学大学院総合科学技術研究科
Graduate School of Integrated Science and Technology, Shizuoka University
†3 静岡大学学術院情報学領域 / JST さきがけ
College of Informatics, Academic Institute, Shizuoka University / JST PRESTO

Vol.2016-MBL-80 No.12
Vol.2016-CDS-17 No.12
2016/8/25



(a) 一般的なセンサネットワークを用いたアーキテクチャ



(b) センサノードをLAN上に配置するアーキテクチャ

図 1 システムアーキテクチャ

いる．またガラスハウス等の施設園芸環境へ応用することで，ハウス内の温湿度のムラの可視化を実現している．しかし，温湿度は単位時間あたりの変化量が比較的小さく，サンプリング周期を秒または分単位で設定することが多い．そのため，温湿度可視化システムでは加速度といった高レートで生成されるストリーミングセンサデータのリアルタイム可視化は想定されていない．

ストリーミングセンサデータを用いた可視化システムとして，加速度データを用いた振動可視化システム[6]がある．振動可視化システムは建造物に設置した WSN から加速度を収集し，建造物の振動を可視化する．歩道橋でのシステムの動作検証にて，床板の振動を可視化することに成功した．しかし，このシステムは，振動計測から可視化までのリアルタイム実行およびウェブブラウザを用いたモニタリングや，インターネットを介した遠隔モニタリングは想定されていない．ユースケースを限定せず汎用的な可視化システムとするには，多くの端末に内蔵されているウェブブラウザを用いて，リアルタイムモニタリングや遠隔モニタリングを可能とすることが望ましい．

3. 空間補間を用いたストリーミングセンサデータ向けリアルタイム可視化システム

3.1 システムアーキテクチャ

本研究では，高レートで生成されるストリーミングセンサデータを，空間補間を用いてリアルタイムに可視化できるシステムを実現するため，データ伝送と可視化処理を低遅延に実行可能な可視化システムを提案する．提案システムのアーキテクチャを**図 1** に示す．システムは，大別してデータ生成部とデータ伝送部，データ可視化部から構成される．データ生成部では多点配置したセンサノードにて環境データのセンシングを行い，センサデータをセンサ GW へ集約する．データ伝送部ではセンサ GW に集約されたセンサデータをクライアント端末へ伝送する．データ可視化部ではクライアント端末にてセンサデータの空間補間と描画をリアルタイムに行う．既存の可視化システムの多くは，センサデータを，DB サーバを介してクライアントへ提供するアーキテクチャを採用しているが，センサ GW から直接クライアントへ伝送することで DB へのアクセスに要する遅延を削減し，センサデータをリアルタイムにクライアントへ提供する．センサデータの蓄積は，システムのクライアントとして DB サーバを設置することで，DB サーバがセンサデータを受信でき，蓄積が可能となる．

図 1 (a)は通信規格として ZigBee 等を用いた一般的なセンサネットワークでのデータ収集を想定したアーキテクチャである．図 1 (b)は Raspberry Pi 等で開発したセンサノードを，Wi-Fi を用いて LAN に接続し，データ収集を行うアーキテクチャである．ZigBee の通信規格である IEEE 802.15.4 と Wi-Fi の通信規格の一つである IEEE 802.11g はともに 2.4 GHz 帯を用いて無線通信を行うが，通信速度と消費電力において異なる性質を有する．通信速度は IEEE 802.15.4 が 250 kbps，IEEE 802.11g が 54 Mbps である．一方，IEEE 802.15.4 は IEEE 802.11g に比べ消費電力が低く，長期間のセンシングに適している．そのため，収集するデータの特性やシステムの利用目的に応じて適切な通信モジュールを採用可能なアーキテクチャとする．

3.2 システムへの要求事項

新たなシステムを社会に導入する際，懸念事項の一つとして導入コストがある．社会に有益なシステムであっても費用対効果が低ければ導入が見送られる．そこで，本システムでは導入コストを抑えるため，クライアント端末ではウェブブラウザを用いて可視化する．ウェブブラウザは PC やスマートデバイスといった汎用端末に標準搭載されている．そのため，ユーザは自身が所持している端末を用いてシステムを利用でき，新たな端末を購入する必要はない．さらに，多くのウェブブラウザが対応している HTML5 の機能を用いて可視化することで，特別なプラグインが不要となり導入時のユーザ負担を軽減することができる．

また可視化可能なストリーミングセンサデータの生成レートは，クライアント端末画面のリフレッシュレートに依存する．一般的な液晶ディスプレイのリフレッシュレートは 60 Hz であることから，ディスプレイの描画性能を最大限利用したシステムとするため，提案システムでは 60 Hz で生成されるストリーミングセンサデータを，フレームレート 60 fps でリアルタイムに可視化できることを目標とする．60 Hz で生成されるストリーミングセンサデータをリアルタイムに可視化するためには，センサ GW 端末からクライアント端末へのデータ伝送時間と，クライアント端末での可視化処理時間の短縮が必要である．データ伝送時間を短縮することでデータ生成から可視化までの遅延を削減でき，可視化のレスポンスが向上する．またクライアン

ト端末での可視化処理時間を短縮することで，空間補間を用いた可視化のように高負荷な可視化処理を要する場合でも，高レートで生成されるストリーミングセンサデータをリアルタイムに可視化可能となる．

3.3 データ伝送時間の短縮

本システムにおけるセンサ GW 端末からクライアント端末へのデータ伝送時間の短縮に関する検討を行う．データ伝送時間を短縮するには，大きく二つの検討事項が存在すると考える．

一つ目の検討事項はデータの伝送方式である．クライアント端末ではウェブブラウザを用いて可視化するため，データ伝送にはウェブブラウザ上で動作する伝送方式を採用する．リアルタイムな可視化を行うためには，ユーザがウェブブラウザで更新ボタンをクリックすることなく，非同期で可視化画面が更新されることが望ましい．また，ストリーミングセンサデータを効率よく伝送するためには軽量な伝送方式である必要がある．ウェブブラウザ上で動作する非同期通信として, Ajax と WebSocket が挙げられる. Ajax は HTTP を用いて動作するため，データ受信のためにはクライアントサイドからのリクエスト送信やパケットへの HTTP ヘッダ付与が必要である．一方，WebSocket はデータをサーバサイドからクライアントサイドへプッシュ送信することができ，クライアントサイドからのリクエスト送信は不要である．またコネクション確立後に送信するパケットにはアプリケーション層のヘッダが不要であることから，WebSocket は Ajax に比べ軽量な伝送方式であるといえる．実際に，Ajax と WebSocket のデータ伝送性能の比較を行った研究[9]では, WebSocket は Ajax に比べネットワークの使用帯域幅が 50%下回ることが示されている. WebSocket は非同期通信が可能で Ajax に比べ軽量な伝送方式であることから，本システムのセンサ GW とクライアント端末のデータ伝送方式として WebSocket を採用する．

二つ目の検討事項は空間補間処理の実行場所である．空間補間処理の実行場所として，センサ GW 端末とクライアント端末の二つがある．クライアント端末への伝送データは，センサ GW で空間補間処理を行う場合は補間済みデータ，クライアント端末で空間補間処理を行う場合は未補間データとなる．未補間データは補間済みデータと比べデータ量が非常に少ないことから，クライアント端末で空間補間処理を行うことで伝送データ量を抑えることができ，データ伝送時間を短縮できる．一方で，センサ GW 端末にて空間補間処理を行うことが望ましい場合も存在する．センサ GW 端末がクライアント端末に比べ高性能であり，補間済みデータの伝送に伴う伝送時間の増分を，センサ GW 端末での高速な空間補間処理で解消できる場合である．この場合，データ伝送時間は増大するが可視化のレスポンスは向上する．しかし，一般にセンサ GW として使用される端末はノートパソコンや Raspberry Pi 等の小型コンピュータである．そのため，センサ GW 端末の性能はクライアント端末であるノートパソコンやスマートデバイスと同程度であると想定できる．そのため，センサ GW 端末とクライアント端末で同じ空間補間アルゴリズムを実行した場合の処理時間には大きな差は生じない．以上より，本システムでは空間補間処理をクライアント端末で実行場所することでデータ伝送時間の短縮を図る．

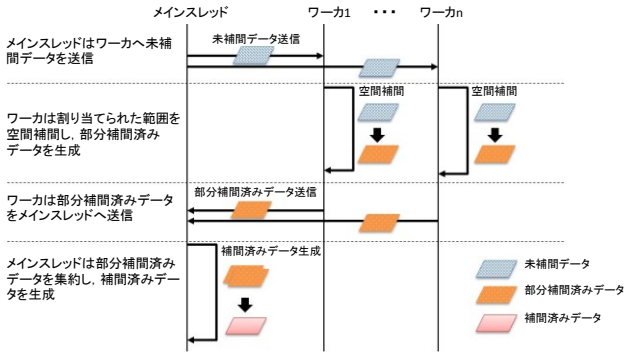


図 2 空間補間の並列処理のシーケンス

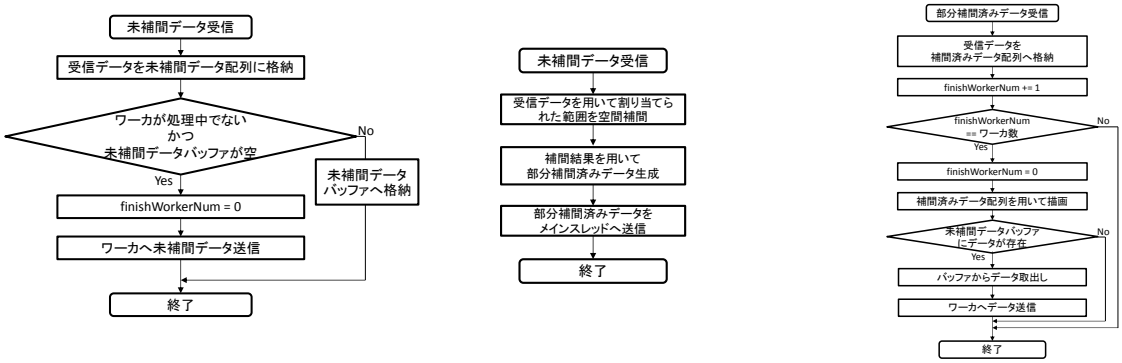
3.4 クライアント端末での可視化処理時間の短縮

クライアント端末では，センサ GW から伝送された未補間データを受信し，可視化処理を行う．可視化処理は，未補間データの空間補間と，補間済みデータの描画に大別される．本システムで想定するデータ生成レートは 60 Hz であるため，クライアント端末へのデータ入来周期は約 16.7 msec である．空間補間と描画の処理が 16.7 msec 以内に完了しない場合，クライアント端末に未処理データが蓄積されリアルタイムな可視化が困難となる．一方，詳細な空間補間を行い高精細な可視化を行う場合，空間補間と描画の処理時間は増大する．高精細な可視化をリアルタイムに実現するためには，空間補間と描画の処理時間を可能な限り短縮することが重要である．そのため，本システムにおける可視化処理時間の短縮手法として WebWorkers を用いた空間補間の並列処理と，WebGL を用いた描画を提案する．

3.4.1 WebWorkers を用いた空間補間の並列処理

WebWorkers とはウェブブラウザで JavaScript の並列処理を可能とする標準仕様である．メインスレッドからワーカと呼ばれるスレッドを起動することで，バックグラウンドでの処理が可能となる. WebWorkers を用いた空間補間の並列処理のシーケンスを**図 2** に示す．メインスレッドが各ワーカへ未補間データを送信すると，各ワーカは割り当てられた範囲の空間補間を行い，部分的な補間済みデータを生成する．各ワーカは生成した部分補間済みデータをメインスレッドへ送信し，メインスレッドにて全ての部分補間済みデータを集約し，補間済みデータを生成する．

図 3 にメインスレッドとワーカの詳細動作を示す．図 3(a)はメインスレッドがセンサ GW から未補間データ



(a) メインスレッドがセンサGWから未補間データを受信した時の動作　(b) ワーカがメインスレッドから未補間データを受信した時の動作　(c) メインスレッドがワーカから部分補間済みデータを受信した時の動作

図 3 WebWorkers を用いた空間補間の並列処理におけるメインスレッドとワーカの動作

を受信した際の動作である．受信時に「ワーカが処理中でない」かつ「未補間データバッファが空である」場合，受信した未補間データをワーカへ送信する.「ワーカが処理中」または「未補間データバッファにデータが存在する」場合はデータを未補間データバッファへ格納する．このバッファに格納されたデータは，全ワーカの処理が完了した後，古いデータから順にワーカへ送信される．図 3(b)はワーカがメインスレッドから未補間データを受信した際の動作である．ワーカは未補間データを受信すると割り当てられた範囲の空間補間を行い，部分補間済みデータを生成する．生成した部分補間済みデータをメインスレッドへ送信し，ワーカの処理は終了する．図 3(c)はメインスレッドがワーカから部分補間済みデータを受信した際の動作である．受信した部分補間済みデータは補間済みデータ配列へ格納する．その後，処理を終えたワーカ数を表す変数 finishWorkerNum をインクリメントする．finishWorkerNum の値が使用しているワーカ数に達し，全ワーカの処理完了を検知すると，補間済みデータ配列を用いて描画を行う．最後に，未補間データバッファにデータが存在する場合はバッファからデータを取り出しワーカへ送信する．

メインスレッドにて未補間データバッファを用意しなくても，ワーカには受信キューが存在する．しかし，受信キューはワーカごとに独立して存在するため，あるワーカが処理を終えた時，他のワーカが処理中であっても，受信キューに存在する次のデータを用いて処理を再開してしまう．そのため，メインスレッド側でデータをバッファリングし，全てのワーカの処理が完了したことを検知した後，各ワーカへ次のデータを送信する．

WebWorkers を用いた並列処理において大規模データの送受信を行う場合は，送受信データに Transferable objects のクラスを用いることが望ましい. Transferable objects とは, WebWorkers においてデータの実体のコピーが伴わない, 高速な送受信が可能なクラスの総称である．ワーカからメインスレッドへ送信する部分補間済みデータは未補間データに比べ大規模なデータであるため, Transferable objects に含まれる Float32Array クラスのオブジェクトとすることで，送受信における遅延の解消を図る．ただし，Transferable objects は一度送信すると送信元からはオブジェクトへアクセスできなくなる．メインスレッドからワーカへの未補間データの送信は，複数のワーカへ同じデータを送信するため，データのコピーを伴う通常の送受信を用いる．

3.4.2 WebGL を用いた描画

本システムの描画形状であるヒートマップは多数の点（四角形）から構成され，可視化にはウェブブラウザを用いる. そのため, 本システムにはウェブブラウザにおいて，多数の点を高速に描画できる描画方法が適していると考える. HTML5 の描画 API である WebGL や Canvas 2D Context を用いて点を描画するには，点の座標・サイズ・色を指定する．ヒートマップ状に繰り返し描画する場合，各点の座標とサイズの情報は更新する必要はなく，最低限更新する必要があるのは色の情報のみである．

Canvas 2D Context で四角形を描画するメソッド fillRect() を用いて繰り返し四角形を描画する際, 点の座標・サイズ・色を毎回指定する必要がある．しかし，WebGL では座標・サイズ・色を独立して設定でき，一度設定した情報は GPU 上のメモリに保持されるため，再描画を行うための処理は色情報の再設定のみとなる．さらに，WebGL では点の座標やサイズ，色の情報から各ピクセルに出力する色の算出を GPU 上で動作するプログラムで行う．そのため，本システムの描画方法として WebGL を用いることで，高速な描画処理が実現できると考える．

4. プロトタイプ実装

4.1 リアルタイム振動可視化システム

第 3 章で述べた本システムのプロトタイプをとして，**図 4** に示すリアルタイム振動可視化システムを開発した．**表 1** に本プロトタイプに用いた端末のハードウェア構成を示す．本プロトタイプでは 3.4 節で述べたクライアント端末での可視化処理時間の短縮手法の有効性検証に重点を

Vol.2016-MBL-80 No.12
Vol.2016-CDS-17 No.12
2016/8/25

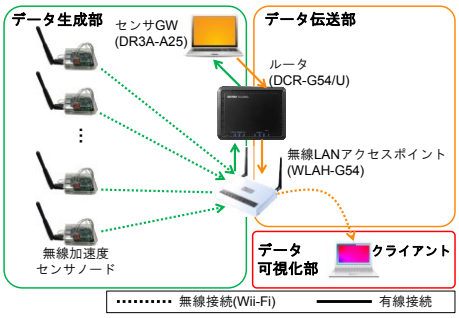


図 4 リアルタイム振動可視化システム

表 1 使用端末のハードウェア構成

	無線加速度センサノード	センサ GW	クライアント
型番(製造元)	Raspberry Pi2 (RaspberryPi Foundation)	DR3A-A25 (ONKYO)	R732/E26GB (TOSHIBA)
OS	Raspbian 8.0	Ubuntu 14.04.4 LTS 64bit	Windows7 Professional 64bit
プロセッサ	ARM Cortex-A7	Intel Core i5-2430M	Intel Core i5-3230M
クロック周波数	900 MHz	2.40 GHz	2.60 GHz
コア数 (論理コア数)	4 (4)	2 (4)	2 (4)
メインメモリ	1 GB	8 GB	8 GB
ストレージ (容量)	microSD (32 GB)	HDD (500 GB)	SSD (256 GB)

置くため，ストリーミングデータを安定して生成できる図 1(b)の Wi-Fi を用いたアーキテクチャを採用した．

4.2 データ生成部の実装

データ生成部は複数の無線加速度センサノードとセンサ GW 端末で構成され，センサノードがセンシングしたデータをセンサ GW 端末に集約する．この時，センサノードからセンサ GW へのデータ伝送においてデータの欠落が生じることが考えられる．TCP 通信を用いて欠落が生じた場合は再送処理を行うことで，高信頼なデータ伝送が可能となる．一方，UDP 通信を用いる場合，再送処理が行われないため，データの欠落が生じるが，常に最新のデータを GW へ送信することができる．データが欠落した場合は，直前のデータで代用することができ，本システムではリアルタイムな振動可視化を目指していることから，常に最新のデータを GW で集約できる UDP 通信を用いることとした．

4.2.1 無線加速度センサノードの実装

無線加速度センサノードの構成を**図 5** に示す．センサノードは加速度センサのセンシングとセンサ GW へのセンサデータ送信を周期的に行う．センサノードへの電源供給に用いたモバイルバッテリ（CHE-061）は，IoT 機器に対応しており，出力電流が小さくなった場合でも電源供給を停止しない．そのため，継続した電源供給を必要とする Raspberry Pi2 を安定して動作させることができると考えた．加速度センサは，Raspberry Pi2 においてセンサデータ読み出し処理の実装が容易な I2C 通信が使用可能であり，比較的安価である ADXL345 を使用した．無線 LAN 子機には小型なチップアンテナに比べ長距離通信が可能なラバーダックアンテナを搭載する LAN-WH300NU2 を採用した．

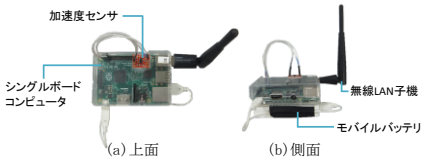


図 5 無線加速度センサノードの構成

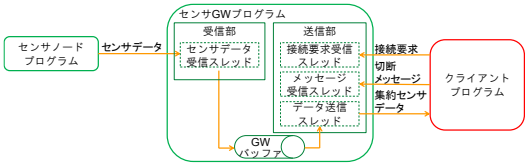


図 6 センサ GW プログラムの構成

4.2.2 センサ GW の実装

センサ GW で動作するセンサ GW プログラムの構成を**図 6** に示す．センサ GW プログラムは受信部と送信部に大別される．受信部のセンサデータ受信スレッドでは，センサノードから送信された加速度データを受信し，GW バッファへ格納する．送信部の接続要求受信スレッドでは，クライアントプログラムからの WebSocket 通信の接続要求の受信を待機し，要求を受信した場合はクライアント追加処理を行う．メッセージ受信スレッドではクライアント端末からの切断メッセージの受信を待機し，切断メッセージを受信した場合，クライアント削除処理を行う．データ送信スレッドでは GW バッファに格納されたデータから未補間データを生成し，クライアントプログラムへ送信する．

受信部と送信部を明確に分離することで，データ収集方法の変更に伴い，データの形式や受信方法に変更が生じる場合でも，受信部の修正のみで変更内容を吸収できる．そのため，プロトタイプの運用時に環境や目的に応じてデータの収集方法を変更する場合にも，センサ GW プログラムの修正は受信部に限定することができる．

4.3 データ伝送部の実装

データ伝送部ではセンサ GW に集約されたデータをクライアント端末へ送信する．センサ GW 端末とクライアント端末の LAN への接続は無線接続と有線接続のいずれかを選択する必要があり，両者には，通信の信頼性と端末の可搬性のトレードオフ関係がある．一般にセンサ GW 端末は固定設置されるため，可搬性よりも通信の信頼性を重視し有線接続とした．クライアント端末はユーザが端末を持ち歩きながらシステムを利用可能とするため，可搬性を重視し無線接続とした．

4.4 データ可視化部の実装

データ可視化部ではクライアント端末のウェブブラウザ上でセンサデータの可視化を行う．クライアント端末での可視化例を**図 7** に示す．空間補間の結果から加速度デー

Vol.2016-MBL-80 No.12
Vol.2016-CDS-17 No.12
2016/8/25

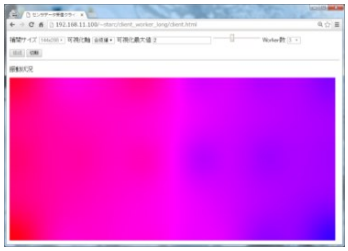
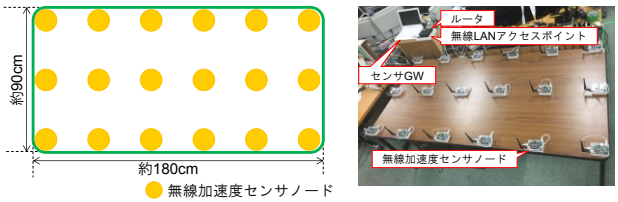


図 7 可視化例



(a) センサノード配置図　(b) 現場環境

図 8 基礎評価実験環境

タの値が高い位置は赤色で，低い位置は青色で表現する．

可視化を実現するクライアントプログラムは，センサGW から未補間データを受信し，空間補間と描画を行う．空間補間アルゴリズムには逆距離加重法（IDW:Inverse Distance Weighting），クリギング法，最近傍補間法等が存在する．クリギング法は高精度な補間が可能であるが，計算量が膨大という欠点がある．最近傍補間法は，アルゴリズムが単純であり計算量が小さいが，補間精度が極めて低い．IDW は計算量がクリギングに比べ少なく，補間性能は最近傍補間に比べ高いという特徴がある．またパラメータチューニングが容易であり実用性の高い補間アルゴリズムであると考え，本プロトタイプでは，IDW を採用した．IDW のアルゴリズムは式(1), (2)で表される．

$$u(x) = \frac{\sum_{i=1}^{N} w_i(x) \cdot (x_i)}{\sum_{j=1}^{N} w_j(x)} \quad (1)$$

$$w_i(x) = \frac{1}{d(x, x_i)^p} \quad (2)$$

Nはデータ数，$u(x)$は点xの値，$d(x, x_i)$は点xと点x_iのユークリッド距離，pは補間値の算出における近傍点の影響度を調整するパラメータである．本プロトタイプにおいてpは一般的な 2 とした．

5. 基礎評価

5.1 実験内容

実装したプロトタイプシステムを用いて提案システムの基礎評価実験を行った．本実験では，3.4 節で述べた可視化処理時間の短縮手法の有効性を検証した．

まず，WebGL を用いた描画の有効性を検証するため，システムの描画方法として WebGL を用いた場合と Canvas 2D Context を用いた場合の描画処理時間の比較を行った．評価に用いた可視化の粒度は 9×18 点，18×36 点，36×72 点，72×144 点，144×288 点の 5 段階を用いた．空間補間の並列処理の評価の前に描画方法の評価を先に行う理由は，空間補間の並列処理の評価にて用いる可視化の粒度を決定するためである．本システムのデータ生成レートは 60 Hz であるためクライアント端末には約 16.7 msec 周期でデータが入来する．描画処理に 16.7 msec 以上要する可視化の粒度では，空間補間の並列処理を組み合わせたとしてもシステムとして遅延のない可視化が困難となる．そのため，描画方法の評価を先に行うことで，空間補間の並列化の評価に用いる可視化の粒度を決定することとした．

次に，WebWorkers を用いた空間補間の並列処理の有効性を検証するために，並列数（ワーカ数）を 1 から 8 まで変化させ，空間補間の処理時間を比較した．並列化の効果はクライアント端末の論理コア数に依存すると考える．並列数を 3 とした場合，クライアントプログラムはメインスレッドが 1，ワーカスレッドが 3 の合計 4 スレッドが動作する．クライアント端末の論理コア数は 4 であることから並列数を 4 以上（5 スレッド以上での動作）としても処理時間は短縮されないと考える．並列数は本仮説を検証するために十分である 1 から 8 とした．

クライアントプログラムの可視化性能を評価するため，空間補間処理時間の計測と同時に，可視化のフレームレートを計測した．クライアント端末へのデータ入来周期はセンサデータの生成周期に等しい．そのため，可視化のフレームレートがセンサデータ生成レートの 60 Hz に一致する場合，クライアント端末では入来するデータを遅延なく可視化できているといえる．

実験環境を**図 8** に示す．本実験は研究室内の机（幅約 180 cm，奥行約 90 cm，高さ約 70 cm）に 18 台の無線加速度センサノードを設置し，システムを動作させる形で実験を行った．空間補間と描画の処理時間とフレームレートの計測値は，1000 回処理を行った際の平均値とした．ウェブブラウザでの描画に用いる HTML の canvas 要素のサイズは，可視化対象である机の縦横比 1:2 に合わせ 450×900 とし，ウェブブラウザは Chrome と Firefox を用いて評価した．

5.2 描画処理時間

描画方法として WebGL を用いた場合と Canvas 2D Context を用いた場合の各描画処理時間を**図 9** に示す．図 9 から WebGL は Firefox での 9×18 点の描画を除く 9 項目で Canvas 2D Context に比べ高速に描画処理を行えることがわかる．WebGL と Canvas 2D Context どちらも可視化の粒度が精細になるにつれて処理時間が増大する．しかし，WebGL では最も精細な可視化の粒度 144×288 点を描画した場合でも，Canvas 2D Context に比べ Chrome では約 163 倍高速な 1.1 msec，Firefox では約 72 倍高速な 1.7 msec で処理を終えることを示した．WebGL を用いた描画では

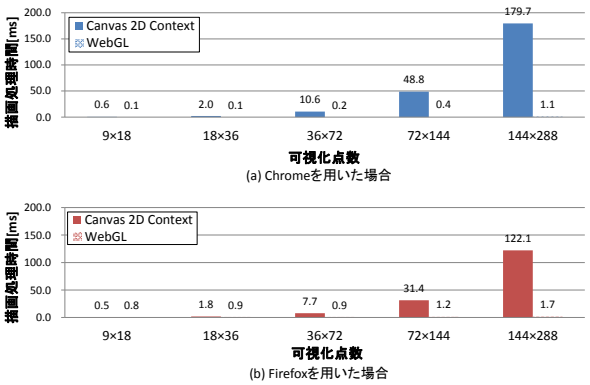


図 9 描画の処理時間

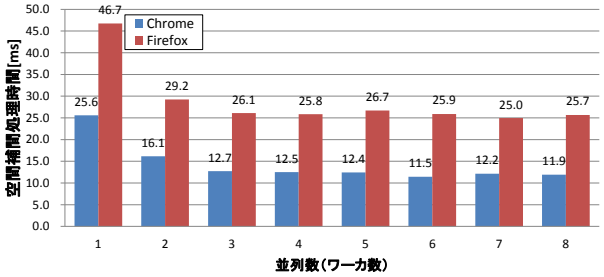


図 10 空間補間の処理時間

Chrome が Firefox に比べ高速に処理を終えたことがわかる．詳細な要因は調査中であるが，要因として JavaScript エンジンの違いが考えられる．JavaScript エンジンとして，Chrome は V8 を，Firefox は SpiderMonkey を採用しており JavaScript エンジンの違いから処理時間の差が生じていると考えられる．

5.3 空間補間処理時間

空間補間をシングルスレッドで処理した場合と並列に処理した場合の空間補間処理時間を**図 10** に示す．描画方法の評価において，WebGL を用いた描画では最も精細な可視化の粒度 144×288 点においても Chrome では 1.1 msec，Firefox では 1.7 msec で描画処理が実行できることが図 9 にて示された．そのため，本評価では可視化の粒度として 144×288 点を用いた．図 10 から空間補間は並列処理化することで処理時間が短縮されることを確認した．並列数を増加させて行くと処理時間の短縮が並列数 3 で頭打ちとなり，並列数 4 以上の場合は大きな性能向上が見られない．5.1 節で述べた，空間補間の並列処理化における処理時間の短縮はクライアント端末の論理コア数に依存するという仮説を裏付ける結果が得られた．また Chrome は Firefox に比べ高速に空間補間処理を行えることが示された．WebGL を用いた描画処理における処理時間の差と同様に，JavaScript エンジンの違いが性能差の要因の一つであると考えられる．

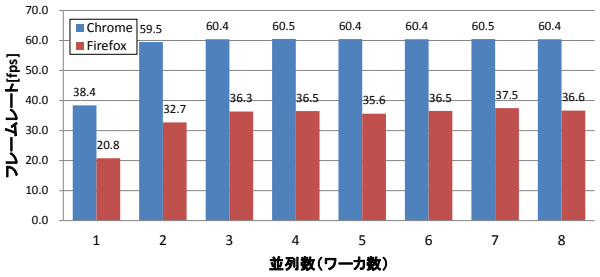
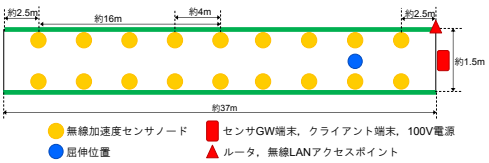


図 11 可視化フレームレート



(a) 使用機器配置と屈伸位置

(b) 現場環境

図 12 実証実験環境

5.4 可視化フレームレート

図 11 に可視化フレームレートの計測結果を示す．図 11 から Chrome を用いた可視化では並列数 3 以上の場合，フレームレートが 60 fps に達しているため 60 Hz で生成されるデータを遅延なく可視化できることが示された．一方で，Firefox を用いた可視化ではフレームレートが 60 fps に満たないことが分かる．この要因として空間補間処理における遅延が挙げられる．描画方法の検証から WebGL を用いることで Firefox でも約 1.7 msec で描画処理が完了することが示された．しかし，空間補間は並列化を用いても処理時間が最小で 25.0 msec とデータ入来周期を上回った．そのため，空間補間処理における遅延が生じ，フレームレートが 60 fps に達しなかったと考える．また，クライアントの処理性能はウェブブラウザだけでなく，クライアント端末のハードウェア性能にも依存すると考えられる．この対策として，可視化の粒度をクライアントの処理性能に適した値に設定することで解決できると考える．可視化の粒度を粗くすると空間補間の処理時間も短縮される．そのためクライアントの処理性能に適した可視化の粒度を設定することで，遅延のない可視化が実現できると考える．可視化開始時に処理性能のベンチマークを行い，ベンチマーク結果から可視化の粒度を決定することで，使用しているウェブブラウザやクライアント端末の性能に適した可視化の粒度を動的に決定できると考える．

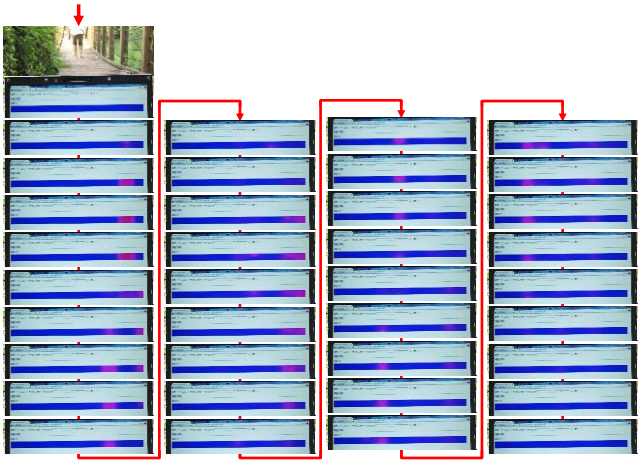


図 13 人間が屈伸を行ったときの可視化結果

6. 屋外評価

6.1 実験内容

本研究で実装したリアルタイム振動可視化システムの有用性を評価するため，システムを静岡県浜松市内の吊橋（橋長約 37 m，幅員約 1.5 m）に設置し吊橋の振動可視化実験を行った．吊橋上に 18 台のセンサノードを配置し，吊橋の上で屈伸を行うことで振動を与え，クライアント端末での可視化を観測した．**図 12** に実験環境を示す．観測においてはクライアント端末の画面を iPhone6 のカメラを用いて 60 fps で動画撮影し，記録した．

6.2 実験結果・考察

撮影した動画からフレーム画像を生成し，時系列に並べた一連の画像を**図 13** に示す． 図 13 から屈伸を行った地点から振動が左右に伝播していることがわかる．振動の伝播は吊橋の構造に関係するが，ウェブブラウザを搭載する汎用端末で振動の伝播をリアルタイムに可視化できたことから，本システムを用いることで，吊橋の振動状況を容易かつリアルタイムに分析可能と考える．

図 12 に示したように，無線 LAN のアクセスポイントは吊橋を支える柱上に設置した．これは当初センサ GW 端末付近に設置していたが，一部のノードとの通信が行えなかったためである．一般に無線アンテナは地面に近い位置に設置した場合，電波が減衰し通信距離が短くなる．そのため，高い位置に無線 LAN アクセスポイントを設置することで全てのノードと通信が可能となったと考える．

7. おわりに

本研究では，空間補間を用いたストリーミングセンサデータ向けリアルタイム可視化システムを提案し，プロトタイプシステムを開発した．クライアント端末における低遅延な可視化処理の実現のため，空間補間には WebWorkers を用いた並列処理を，描画方法には WebGL を採用した．基礎評価の結果，空間補間は並列処理を用いて処理時間が短縮可能であり，WebGL は Canvas 2D Context に比べ高速な描画処理が行えることを確認した．可視化のフレームレートを計測したところ，空間補間を並列処理し，WebGL を用いて描画することで，60 Hz で生成されるセンサデータを 60 fps でリアルタイムに可視化できることを示した．また，屋外評価の結果，吊橋の振動伝播をリアルタイムに可視化でき，本システムを用いて建造物の振動をリアルタイムに分析できる見通しを得た．

今後の課題としては，インターネットを介したデータ伝送を行う場合やクライアント端末数を増加させた場合の可視化性能の評価があげられる．また，本システムで可視化可能なデータの生成周期はクライアント端末画面のリフレッシュレートに依存する．そのため，一般的な液晶ディスプレイのリフレッシュレートである 60 Hz よりも高いサンプリングレートが必要な振動や音声の可視化は困難である．その対策として 60 Hz よりも高い周波数でサンプリングしたセンサデータを，可視化用に 60 Hz に圧縮するアルゴリズムの検討が必要である．

謝辞 本研究は，半導体理工学研究センター（STARC）共同研究 IS プログラム No.1529 の支援を受け実施したものである．

参考文献

[1] Villanueva FJ, Aguirre C, Rubio AB, Villa D, Santomfiia MJ, Lopez JC: Data stream visualization framework for smart cities, Soft Computing, Volume 20, Issue 5, pp.1671-1681, 2016.
[2] 中根傑，江田政聡，横山昌平，福田直樹，石川博：センサネットワークにおける大規模な可視化システムの開発，第 2 回データ工学と情報マネジメントに関するフォーラム（DEIM2010）講演論文集, 2010.
[3] 伊藤昌毅，片桐由希子，石川幹子，徳田英幸：Airy Notes:緑地計画のための無線センサネットワークによる環境モニタリング，情報処理学会論文誌, Vol. 49, No. 1, pp.69‐82, 2008.
[4] 柴田瞬，丸島晃明，峰野博史，: 3 次元可視化システムと WSN を用いた視覚的評価手法の提案， 第 77 回全国大会講演論文集, Vol.2015， No.1, pp.235-236, 2015.
[5] 松野 智明，串岡 聡，今原 淳吾，福田宗弘，水野忠則，峰野博史：観測データの空間補間を利用した施設園芸環境の可視化・制御システムの提案，マルチメディア，分散，協調とモバイル（DICOMO）シンポジウム論文集, pp.2129-2136, 2012.
[6] 川原正人，中畑和之，大賀水田生:多点同時計測による橋梁床板の動的挙動の 3 次元可視化と歩道橋における実験的検証，構造工学論文集, Vol.59A, pp.1170-1178, 2013.
[7] Ian Hickson: Web Workers, https://www.w3.org/TR/2015/WD-workers-20150924/, (参照 2016/07/20).
[8] Dean Jackson, Jeff Gibert: WebGL Specification, https://www.khronos.org/registry/webgl/specs/latest/1.0/, (参照 2016/05/20).
[9] Puranik, D., Feiock, D., and Hill, J: Real-time monitoring using ajax and websockets, Proceedings of the 20th Annual IEEE International Conference and Workshops on the Engineering of Computer Based Systems, pp.110-118, 2013.