(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第6463914号
(P6463914)

(45)発行日　平成31年2月6日(2019.2.6)　(24)登録日　平成31年1月11日(2019.1.11)

(51)Int.Cl.			F I		
H04N	1/393	(2006.01)	H04N	1/393	
G06T	3/40	(2006.01)	G06T	3/40	
B41J	21/00	(2006.01)	B41J	21/00	Z
B41J	5/30	(2006.01)	B41J	5/30	Z

請求項の数 19　(全 39 頁)

(21)出願番号　特願2014-135179(P2014-135179)
(22)出願日　平成26年6月30日(2014.6.30)
(65)公開番号　特開2016-15538(P2016-15538A)
(43)公開日　平成28年1月28日(2016.1.28)
審査請求日　平成29年6月30日(2017.6.30)

(73)特許権者　000001007
キヤノン株式会社
東京都大田区下丸子３丁目３０番２号
(74)代理人　100076428
弁理士　大塚　康徳
(74)代理人　100112508
弁理士　高柳　司郎
(74)代理人　100115071
弁理士　大塚　康弘
(74)代理人　100116894
弁理士　木村　秀二
(74)代理人　100130409
弁理士　下山　治
(74)代理人　100134175
弁理士　永川　行光

最終頁に続く

(54)【発明の名称】情報処理装置、処理方法、及びプログラム

(57)【特許請求の範囲】
【請求項１】
プロセッサで翻訳され実行されるための命令セットを含む第１のプログラム層と、前記プロセッサ以外で予め翻訳された命令セットを含む第２のプログラム層とを包含するプログラムを、前記プロセッサで実行する情報処理装置であって、
画像データに対して分割処理と拡大処理を実行する分割拡大手段と、
前記分割拡大手段による分割処理と拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行する出力手段とを有し、
前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、
前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、
前記第１のプログラム層では画像データをテキストデータとして扱い、
前記第２のプログラム層では前記画像データをバイナリデータとして扱うことを特徴とする情報処理装置。
【請求項２】
前記出力手段は、前記複数の画像データに基づいて１つの拡大画像データを生成し、前記１つの拡大画像データに基づく印刷データを前記外部デバイスに送信することを特徴とする請求項１に記載の情報処理装置。
【請求項３】

前記出力手段は、前記複数の画像データを順次、前記外部デバイスに送信することを特徴とする請求項１に記載の情報処理装置。
【請求項４】
前記画像データの描画結果を表示する表示手段をさらに有することを特徴とする請求項１乃至３のいずれか１項に記載の情報処理装置。
【請求項５】
前記分割拡大手段により得られた前記複数の画像データがハードディスクまたはフラッシュメモリに記憶されることを特徴とする請求項１乃至４のいずれか１項に記載の情報処理装置。
【請求項６】
前記画像データに付加される付加画像は、前記第１のプログラム層により拡大されることを特徴とする請求項１乃至５のいずれか１項に記載の情報処理装置。
【請求項７】
前記分割拡大手段は、
前記外部デバイスとしてのプリンタの解像度に基いて、画像の拡大率を決定する第１の決定手段と、
前記プリンタが備えるメモリの利用可能な容量と前記情報処理装置の記憶手段に確保が可能な容量とに基いて、前記画像の分割数を決定する第２の決定手段とを含むことを特徴とする請求項１乃至６のいずれか１項に記載の情報処理装置。
【請求項８】
ユーザから前記画像データに対する画像処理の指示を入力する入力手段と、
前記入力手段により入力された画像処理の指示に従って画像処理を行う画像処理手段とをさらに有することを特徴とする請求項１乃至７のいずれか１項に記載の情報処理装置。
【請求項９】
前記画像処理手段による画像処理によりベクターデータにより表現される画像が元の画像に対して付加されることを特徴とする請求項８に記載の情報処理装置。
【請求項１０】
プロセッサで翻訳され実行されるための命令セットを含む第１のプログラム層と、前記プロセッサ以外で予め翻訳された命令セットを含む第２のプログラム層とを包含するプログラムを、前記プロセッサで実行する情報処理装置の処理方法であって、
画像データに対して分割処理と拡大処理を実行し、
前記分割処理と前記拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行し、
前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、
前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、
前記第１のプログラム層では画像データをテキストデータとして扱い、
前記第２のプログラム層では前記画像データをバイナリデータとして扱うことを特徴とする処理方法。
【請求項１１】
ウェブ標準言語で記述された第１のプログラム層と、ウェブ標準言語とは異なるプログラム言語で記述された第２のプログラム層とを包含するプログラムを実行するコンピュータを、画像データに対して分割処理と拡大処理を実行する分割拡大手段と、前記分割拡大手段による分割処理と拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行する出力手段として機能させ、
前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、
前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、

前記第１のプログラム層では画像データをテキストデータとして扱い、
前記第２のプログラム層では前記画像データをバイナリデータとして扱うことを特徴とする前記コンピュータが読み取り可能なプログラム。
【請求項１２】
前記出力手段は、前記複数の画像データに基づいて１つの拡大画像データを生成し、前記１つの拡大画像データに基づく印刷データを前記外部デバイスに送信することを特徴とする請求項１１に記載のプログラム。
【請求項１３】
前記出力手段は、前記複数の画像データを順次、前記外部デバイスに送信することを特徴とする請求項１１に記載のプログラム。
【請求項１４】
前記画像データの描画結果を表示する表示手段として前記コンピュータをさらに機能させるための請求項１１乃至１３のいずれか１項に記載のプログラム。
【請求項１５】
前記分割拡大手段により得られた前記複数の画像データはハードディスクまたはフラッシュメモリに記憶されることを特徴とする請求項１１乃至１４のいずれか１項に記載のプログラム。
【請求項１６】
前記画像データに付加される付加画像は、前記第１のプログラム層により拡大されることを特徴とする請求項１１乃至１５のいずれか１項に記載のプログラム。
【請求項１７】
前記外部デバイスとしてのプリンタの解像度に基いて、画像の拡大率を決定する第１の決定手段と、
前記プリンタが備えるメモリの利用可能な容量と前記コンピュータの記憶手段に確保が可能な容量とに基いて、前記画像の分割数を決定する第２の決定手段として前記コンピュータをさらに機能させることを特徴とする請求項１１乃至１６のいずれか１項に記載のプログラム。
【請求項１８】
ユーザから前記画像データに対する画像処理の指示を入力する入力手段と、
前記入力手段により入力された画像処理の指示に従って画像処理を行う画像処理手段として前記コンピュータをさらに機能させることを特徴とする請求項１１乃至１７のいずれか１項に記載のプログラム。
【請求項１９】
前記画像処理手段による画像処理によりベクターデータにより表現される画像が元の画像に対して付加されることを特徴とする請求項１８に記載のプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、例えば、プリンタなどの画像出力装置に対して外部デバイスとして動作する移動体端末などの情報処理装置で実行される、例えば、レンダリング処理に関するものである。
【背景技術】
【０００２】
近年、カメラ機能が搭載された可搬型多機能携帯端末（以下、モバイルコンピュータ）が爆発的に普及し、デジタルカメラや従来のパーソナルコンピュータ（以下、ＰＣ）の販売台数を遥かに凌ぐ勢いで拡販されている。
【０００３】
このようなモバイルコンピュータは、基本的には３つの要素で成り立っている。即ち、コンピュータ自身であるハードウェアと、該ハードウェア上で動作するオペレーティングシステム（以下、ＯＳ）と、そのＯＳ上で動作するアプリケーションである。ユーザは、

そのアプリケーションを用いて、地図やメール、インターネット上のウェブサイトの閲覧等の機能を使用することが可能である。

【０００４】

このようなモバイルコンピュータ上で動作するアプリケーションの形態としては、主に二つのものが存在する。即ち、ネイティブアプリケーションとウェブアプリケーションである。以下、それぞれの特徴を説明する。

【０００５】

まず、ネイティブアプリケーションとは、通常、ＯＳ毎に用意される開発環境、及び開発言語を用いて開発される。例えば、Ａ社が提供するＯＳ上ではＣ／Ｃ＋＋言語、Ｂ社が提供するＯＳ上ではＪａｖａ（登録商標）言語、Ｃ社が提供するＯＳ上では更に異なる開発言語を用いる、という具合である。通常、ネイティブアプリケーションは、各開発環境において予めコンパイルされ、人間が理解可能ないわゆる高水準言語から、コンピュータのＣＰＵが解釈可能なアセンブラ等の命令セット群に変換される。このように、通常のネイティブアプリケーションでは、命令をＣＰＵが直接解釈するために、高速動作が可能である、というメリットがある。

【０００６】

一方、ウェブアプリケーションとは、近年では、各コンピュータ上のＯＳに標準的に組み込まれているウェブブラウザ上で動作するアプリケーションのことである。そのアプリケーションはウェブブラウザが解釈できるよう、一般的には、ＨＴＭＬ５及びＣＳＳ、さらにＪａｖａＳｃｒｉｐｔ（登録商標）等の言語を用いて開発される。これらはウェブ標準言語であるため、これらのウェブ標準言語でウェブアプリケーションを一旦記述すれば、ウェブブラウザが動作する環境であれば、どこでも動作可能というメリットがある。

【先行技術文献】

【特許文献】

【０００７】

【特許文献１】特開２０１１－２３３０３４号公報

【発明の概要】

【発明が解決しようとする課題】

【０００８】

上述した近年のモバイルコンピュータは、高解像度のカメラを内蔵している。モバイルコンピュータはユーザによって日常携帯されており、かつ写真を数千枚程度記憶可能なメモリを備えているためユーザは非常に高い頻度で気楽に写真撮影を楽しむことができる。このようにして得られた写真画像に対して、例えば、モノクロ・セピア調にするフィルター処理を施したり、写真が暗い、色バランスが崩れている等の不具合を修正するための画像処理は、ユーザにとっても大変重要で欠かせない機能となっている。このような画像処理をユーザが、ストレスなく簡単に実行できるようなアプリケーションを、上記２つの形態で開発する場合、以下のような課題がある。

【０００９】

まずネイティブアプリケーションでは、上述したように処理を高速実行できるというメリットがある。しかし、このネイティブアプリケーションはＯＳ毎に異なる開発言語で別々に開発する必要があるため、開発コストと開発時間が増大し、ユーザに迅速に提供できないという課題がある。また、ネイティブアプリケーションは、予めコンパイル（翻訳）する必要がある。そのため、例えば、アプリケーションのＵＩ設計を動作時に変更したり、機能を動的に追加することが難しく、柔軟性に欠ける。

【００１０】

特許文献１はウェブアプリケーションの形態の一例を開示している。ウェブアプリケーションでは通常、ＨＴＭＬ５、ＣＳＳ、ＪａｖａＳｃｒｉｐｔ（登録商標）で記述されたアプリケーションの本体が、モバイルコンピュータ外部のサーバ上に存在する。そのアプリケーションは、利用時に動的にサーバからモバイルコンピュータにインターネットを介してダウンロードされるため、ＵＩ設計などを予めコンパイルする事なく動的に変更する

ことが可能である。しかしながら、高度で複雑な処理を実行する場合、ウェブアプリケーションは、ブラウザのセキュリティ上の制約からブラウザ上でＪａｖａＳｃｒｉｐｔ（登録商標）によって実行するか、サーバ上で実行するという２つの選択肢しか存在しない。ＪａｖａＳｃｒｉｐｔ（登録商標）は従来より、人間が視認可能な文字列のスクリプトとして記述され、そのスクリプトを動作時に随時コンパイルすることで実行することができるようになっている。このことから、複雑な処理をＪａｖａＳｃｒｉｐｔ（登録商標）で記述するとその動作が遅いという問題がある。

【００１１】

また、その処理をサーバで実行するように構築した場合、モバイルコンピュータ内部に存在する写真などのデータを、インターネットを介してサーバ上にアップロードし、処理後の結果を今度はダウンロードする時間が必要となる。これは、モバイルアプリケーションに対してストレスの少ない、即時的な処理を欲しているユーザにとっては、大きな問題となる。それに加え、サーバでの処理はオフラインでは実行できないという問題がある。

【００１２】

さらに、画像処理において画像拡大処理を実行する場合、メモリ上に拡大前と拡大後の画素の値を保持する必要があるため、画素数が多い程、メモリを多く使用する。しかしながら、上記のようなアプリケーションを、例えばモバイルコンピュータが実行する場合、大容量のＲＡＭをメモリとして実装することは難しい。加えて、上記のようなウェブアプリケーションは画像データをバイナリデータではなく文字列データとして保持するので、同じ量の画像データをバイナリデータで保持する場合と比べて大きな容量のメモリを必要とする。そのため、画像拡大処理には更に大きな容量のメモリを必要とし、コンピュータがメモリ不足に陥りやすい。

【００１３】

本発明は上記の課題を解決するためになされたものであり、ハイブリッドアプリケーションを実行する場合にもメモリ不足を生じさせず、効率的に画像データを処理することが可能な情報処理装置、処理方法、及びプログラムを提供することを目的とする。

【課題を解決するための手段】

【００１４】

上記の目的を達成するための本発明の情報処理装置は以下のような構成を備える。

【００１５】

即ち、プロセッサで翻訳され実行されるための命令セットを含む第１のプログラム層と、前記プロセッサ以外で予め翻訳された命令セットを含む第２のプログラム層とを包含するプログラムを、前記プロセッサで実行する情報処理装置であって、画像データに対して分割処理と拡大処理を実行する分割拡大手段と、前記分割拡大手段による分割処理と拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行する出力手段とを有し、前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、前記第１のプログラム層では画像データをテキストデータとして扱い、前記第２のプログラム層では前記画像データをバイナリデータとして扱うことを特徴とする。

【００１６】

また本発明を別の側面から見れば、プロセッサで翻訳され実行されるための命令セットを含む第１のプログラム層と、前記プロセッサ以外で予め翻訳された命令セットを含む第２のプログラム層とを包含するプログラムを、前記プロセッサで実行する情報処理装置の処理方法であって、画像データに対して分割処理と拡大処理を実行し、前記分割処理と前記拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行し、前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、前記第１のプログラム層では画像データをテキストデータとして扱い、前記第２のプログラム層では前記画像デー

タをバイナリデータとして扱うことを特徴とする処理方法を備える。
【００１７】
さらに本発明を別の側面から見れば、ウェブ標準言語で記述された第１のプログラム層と、ウェブ標準言語とは異なるプログラム言語で記述された第２のプログラム層とを包含するプログラムを実行するコンピュータを、画像データに対して分割処理と拡大処理を実行する分割拡大手段と、前記分割拡大手段による分割処理と拡大処理とにより得られた複数の画像データを用いて外部デバイスに対する出力処理を実行する出力手段として機能させ、前記画像データに対する描画処理は、Ｃａｎｖａｓ又はＳｃａｌａｂｌｅ　Ｖｅｃｔｏｒ　Ｇｒａｐｈｉｃｓ（ＳＶＧ）を用い、前記第１のプログラム層の命令セットは、ＨＴＭＬ、ＣＳＳ、ＪａｖａＳｃｒｉｐｔのいずれかを含み、前記第１のプログラム層では画像データをテキストデータとして扱い、前記第２のプログラム層では前記画像データをバイナリデータとして扱うことを特徴とする前記コンピュータが読み取り可能なプログラムを備える。
【発明の効果】
【００１８】
従って、本発明によれば、例えば、画像拡大処理のような大きな容量のメモリを用いる画像処理を実行する場合にもメモリ不足の発生を抑えることができるという効果がある。
【図面の簡単な説明】
【００１９】
【図１】本発明の代表的な実施形態である情報処理装置の構成を示すブロック図である。
【図２】図１に示した情報処理装置のソフトウェアの構成を示すブロック図である。
【図３】ユーザ操作に伴う処理を示すフローチャートである。
【図４】実施形態１に従う写真画像選択の詳細を示すフローチャートである。
【図５】実施形態１に従う画像処理の詳細を示すフローチャートである。
【図６】実施形態１に従うスタンプ追加の詳細を示すフローチャートである。
【図７】スタンプ特定の詳細を示すフローチャートである。
【図８】スタンプ操作の詳細を示すフローチャートである。
【図９】プリンタ設定の詳細を示すフローチャートである。
【図１０】実施形態１に従うレンダリングの詳細を示すフローチャートである。
【図１１】プリントの詳細を示すフローチャートである。
【図１２】アプリケーション画面の一例を示す図である。
【図１３】設定画面の一例を示す図である。
【図１４】バンドレンダリングを概念的に示す模式図である。
【図１５】実施形態１に従うバンドレンダリングの詳細を示すフローチャートである。
【図１６】実施形態２に従うレンダリングの詳細を示すフローチャートである。
【図１７】実施形態２に従うバンドレンダリングの詳細を示すフローチャートである。
【発明を実施するための形態】
【００２０】
以下本発明の実施の形態について添付図面を参照して詳細に説明する。特に、以下の実施形態では、情報処理装置上で、後述するハイブリッド型写真印刷アプリケーションを動作させ、ユーザが選択した画像に対して、様々な画像処理を適用した後に、その画像を印刷する構成について説明する。なお、その情報処理装置の代表例としては、携帯電話やスマートフォンやタブレット端末のような携帯型情報端末が含まれる。
【００２１】
＜ハードウェア構成の説明＞
図１は本発明の代表的な実施形態である情報処理装置１１５として、例えば、スマートフォンや携帯電話等の携帯型情報端末の構成例を説明するブロック図である。同図において、１００はＣＰＵ（中央演算装置／プロセッサ）であり、以下で説明する各種処理をプログラムに従って実行する。図中のＣＰＵ１００は１つであるが、複数のＣＰＵあるいはＣＰＵコアによって構成されていても良い。１０１はＲＯＭであり、ＣＰＵ１００により

実行されるプログラムが記憶されている。１０２はＲＡＭであり、ＣＰＵ１００によるプログラムの実行時に、各種情報を一時的に記憶するためのメモリである。なお、図１において、ＣＰＵは１つの例が示されているが、複数のＣＰＵを用いる構成、即ち、マルチプロセッサ構成でも良い。

【００２２】

１０３はハードディスクやフラッシュメモリ等の２次記憶装置であり、ファイルや画像解析等の処理結果を保持するデータベース等のデータや、各種プログラムを記憶するための記憶媒体である。１０４はディスプレイであり、各種処理を実現するための操作を受け付けるためのＵＩ（ユーザインタフェース）や、実行された処理による処理結果等の各種情報を表示する。ディスプレイ１０４は、タッチセンサ１０５を備えても良い。

【００２３】

情報処理装置１１５は、内部撮像デバイス１１０を備えてもよい。内部撮像デバイス１１０による撮像によって得られた画像データは、所定の画像処理を経た後、２次記憶装置１０３に保存される。また、画像データは、外部Ｉ／Ｆ（インタフェース）１０８を介して接続された外部撮像デバイス１１１から読み込むこともできる。

【００２４】

情報処理装置１１５は、外部Ｉ／Ｆ（インタフェース）１０９を備え、インターネット等のネットワーク１１３を介して通信を行うことができる。情報処理装置１１５は、この通信Ｉ／Ｆ１０９を介して、ネットワーク１１３に接続されたサーバ１１４より画像データを取得することもできる。

【００２５】

情報処理装置１１５は、加速度センサ１０６を備え、情報処理装置１１５自身の位置姿勢に関する加速度情報を取得することができる。情報処理装置１１５は、外部Ｉ／Ｆ１０７を介し、プリンタ１１２と接続されており、画像データ等のデータを出力することができる。プリンタ１１２は、ネットワーク１１３にも接続されており、通信Ｉ／Ｆ１０９経由で、画像データを送受信することができる。

【００２６】

外部Ｉ／Ｆ１０７～１０９は、有線通信と無線通信の内、少なくともいずれかの通信形態を有するインタフェースであり、利用する通信形態に応じて外部デバイス（プリンタ１１２あるいはサーバ１１４）との通信を行う。有線通信には、例えば、ＵＳＢ、イーサネット（登録商標）等があり、無線通信には、無線ＬＡＮ、ＮＦＣ、Ｂｌｕｅｔｏｏｔｈ（登録商標）、赤外線通信等がある。また、無線通信として、無線ＬＡＮを利用する場合には、装置同士が直接接続する形態もあれば、無線ＬＡＮルータ等の中継装置を介して接続する形態もある。また、外部Ｉ／Ｆ１０７～１０９は、図では別々に構成されているが、一体となって構成されていても良い。

【００２７】

情報処理装置１１５の動作に必要な電源は、バッテリ１１７によって供給される。情報処理装置１１５が備える各種構成要素は、制御バス／データバス１１６を介して相互に接続され、ＣＰＵ１００は、この制御バス／データバス１１６を介して、各種構成要素を制御する。

【００２８】

尚、本実施形態では、情報処理装置１１５が、情報処理装置１１５が備える制御部（ＣＰＵ１００）によって実行されるプログラム等のソフトウェアの実行場所（ソフトウェア実行環境）となる。

【００２９】

＜ソフトウェアのブロック図＞

図２は情報処理装置１１５で動作するソフトウェア構成のブロック図である。

【００３０】

情報処理装置１１５は、スクリプト層２１７、ネイティブ層２１８、及びＯＳ層２１９のプログラムを実行する。これらの各層は、ＣＰＵ１００がＲＯＭ１０１あるいは２次記

憶装置１０３に記憶されている対応するプログラムを読み出し実行することにより実現される。

【００３１】

スクリプト層２１７は、ＨＴＭＬ５やＣＳＳ３、及びＪａｖａＳｃｒｉｐｔ等のウェブ標準言語を使って、テキストデータで命令セット（コンテンツの描画や画像の表示、動画の再生等）が記述されているプログラム層である。スクリプト層２１７では、アプリケーション実行環境上で、そのアプリケーション実行環境に存在するプロセッサ（例えば、ＣＰＵ１００）を用いて、テキストデータの各種命令セットを翻訳して実行することになる。その実行形態としては、実行の度に命令文を一行ずつ動的に翻訳する場合や、アプリケーションを起動したときに翻訳する場合、アプリケーションを情報処理装置１１５にインストールしたときに翻訳する場合等が考えられる。

【００３２】

以後、スクリプト層２１７で処理することや内容をスクリプトと呼ぶ。スクリプトの命令を情報処理装置１１５内で翻訳する形態の例として、ネイティブ層２１８やＯＳ層２１９が備えるインタプリタの機能を使用することが挙げられる。尚、この実施形態においては、アプリケーションのＵＩの大部分が、スクリプト層２１７で記述されていることを想定している。

【００３３】

ネイティブ層２１８は、アプリケーション実行環境以外で予め翻訳（コンパイル）された命令セットが記述されているプログラム層である。形態としては、ＣもしくはＣ＋＋といった高水準言語で記述されたコードが、予めアプリケーションの開発者のＰＣやサーバ上でコンパイルされ、ＣＰＵ１００が解釈可能な命令の集合体となっている。以後、ネイティブ層２１８で処理することや内容、後述するＯＳ層２１９の機能をネイティブ層２１８から呼び出すことを含め、ネイティブと呼ぶこととする。尚、ネイティブ層２１８の別の実装系として、Ｊａｖａが挙げられる。Ｊａｖａは、Ｃ／Ｃ＋＋と類似の高水準言語であり、予めアプリケーション開発時の開発環境上で中間コードに翻訳される。翻訳された中間コードは、各ＯＳが備えるＪａｖａ仮想環境上で動作する。本実施形態においては、このようなプログラム形態も、ネイティブ層２１８の一種に含める。

【００３４】

ＯＳ層２１９は、情報処理装置１１５のオペレーティングシステム（Ｏｐｅｒａｔｉｎｇ　Ｓｙｓｔｅｍ：ＯＳ）に対応する。ＯＳ層２１９は、ハードウェア機能の使用をアプリケーションに提供する役割及び固有の機能を有する。ＯＳ層２１９は、ＡＰＩを備え、スクリプト層２１７やネイティブ層２１８から機能を使用することができる。

【００３５】

この実施形態では、スクリプト層２１７からネイティブ層２１８の呼び出しを可能にすることをバインディング、もしくはバインドと呼ぶ。各種ネイティブの機能は、ＡＰＩを備え、該ＡＰＩをスクリプトが呼び出すことでネイティブの機能を使用することができる。このようなバインディング機能は、通常、各種ＯＳが標準的に備えている機能である。

【００３６】

尚、この実施形態では、スクリプト層２１７とネイティブ層２１８を含むアプリケーションのことをハイブリッドアプリケーションと呼ぶ。

【００３７】

スクリプト層２１７の画像取得部２０１は、ネイティブ層２１８に対し画像データの取得を依頼する。取得依頼時に、画像取得部２０１は、一意なＩＤを生成し、ネイティブ層２１８に送信する。このＩＤと、ネイティブ層２１８の画像読込部２０２で読み込まれた画像データは、対となって、ネイティブ層２１８のデータ保持部２０４に記憶される。これ以外にも、例えば、絶対パスを指定する方法や、ダイアログ表示を促す方法等が取得依頼の方法として挙げられる。

【００３８】

ネイティブ層２１８の画像読込部２０２は、画像データ群２１５から画像データを取得

する。画像データ群２１５からの画像データの取得方法は、スクリプト層２１７の画像取得部２０１の依頼に依存する。依頼方法は、ＵＩ上に提供されているダイアログボックスから選択する、ファイルのパスから直接画像を選択する等が挙げられる。

【００３９】

ネイティブ層２１８のデータ変換部２０３は、ネイティブ層２１８のデータ（例：バイナリ形式の画像データ）をスクリプト層２１７で利用できる形式のデータ（例：テキスト形式（ＢＡＳＥ６４）の画像データ）に変換する。一方で、データ変換部２０３は、スクリプト層２１７から送られてきたデータ（例：テキスト形式（ＢＡＳＥ６４）の画像データ）をネイティブ層２１８で利用できる形式のデータ（例：バイナリ形式の画像データ）にする変換も行う。

【００４０】

スクリプト層２１７のデータ変換部２０７は、スクリプト層２１７のデータ（例：テキスト形式の処理パラメータ）をネイティブ層２１８で利用できる形式のデータ（例：テキスト形式（ＪＳＯＮ形式）の処理パラメータ）に変換する。

【００４１】

ネイティブ層２１８のデータ保持部２０４は、画像読込部２０２で読み込んだ画像データ、画像処理部２０８で画像処理が施された画像データを保持する。保持される画像データは、例えば、ＲＧＢ画像データに展開されており、すぐに画像処理が実行できる形式になっている。また、保持されている画像データは画像取得部２０１で生成したＩＤと対になっており、データ保持部２０４から画像データを取得する際はＩＤを指定すればよい。

【００４２】

スクリプト層２１７のコンテンツ描画部２０５は、プリントのためのコンテンツをウェブ標準言語を利用して記述する。この記述には、コンテンツ操作部２１０で操作されたスクリプトも反映される。コンテンツ描画部２０５で記述されたコンテンツのスクリプトは、ＯＳ層２１９のインタプリタ２１４で解釈され、ディスプレイ１０４に表示される。

【００４３】

スクリプト層２１７の画像処理制御部２０６は、画像処理に用いる補正パラメータと、処理対象となる画像を決定し、画像処理部２０８に画像処理を依頼する。補正パラメータはデータ変換部２０７でネイティブ層へ送信できる形式へ変換された後、処理対象となる画像のＩＤと共にネイティブ層へ送信される。

【００４４】

ネイティブ層の画像処理部２０８は、画像処理制御部２０６で指定された画像に対し画像処理を施す。その際、どのような画像処理を施すかは、画像処理制御部２０６で設定されたパラメータにより決定される。画像の指定については、例えば、スクリプト層から画像のパスを受け取る方法や、画像データごと受け取る方法などが考えられる。

【００４５】

ＯＳ層２１９のタッチイベント２０９は、ディスプレイ１０４のタッチに関する情報を取得する。タッチに関する情報とは、ディスプレイ１０４のタッチ検知、タッチされた位置情報等が挙げられる。取得したタッチに関する情報は、ネイティブ層２１８経由でスクリプト層２１７のコンテンツ操作部２１０に送信される。

【００４６】

スクリプト層２１７のコンテンツ操作部２１０は、画像を操作、例えば、画像の拡大、移動、回転などを行い、その操作を反映すべく、スクリプト命令を変更する。

【００４７】

スクリプト層２１７のプリンタ制御部２１１は、レンダリング部２１６へのレンダリング開始依頼、プリンタ検知の依頼、プリンタ設定画面の表示、プリント情報の生成と送信を制御する。プリンタ設定画面では、用紙のサイズ、用紙の種類、カラー・モノクロ等のプリンタ設定がなされる。ここで設定された項目を基に、プリンタデータ生成部２１２でプリンタデータが生成される。

【００４８】

ネイティブ層２１８のプリンタデータ生成部２１２は、プリンタ制御部２１１からの依頼を基に、プリンタ通信に必要なデータ、コマンドを生成する。プリンタ通信に必要なデータとは、通信プロトコルに則ったデータであり、コマンドとは、印刷やスキャン等、プリンタの動作を決定するためのデータである。よって、プリンタデータ生成部２１２は、プリンタの動作を決定するためのコマンドを含むプリンタデータを生成する。

【００４９】

さて、ネイティブ層２１８の外部デバイス通信部２２１は、プリンタなどの情報処理装置１１５と接続している外部デバイスとの通信を行うためのインタフェース（ＩＦ）である。ここでは、プリンタデータ生成部２１２から受け取ったデータを送信したり、プリンタ１１２からの情報を受信する。この実施形態ではＯＳ層２１９の通信制御部２１３を介して通信するが、外部デバイス通信部２２１が直接、外部ＩＦ１０７へデータを送信してもよい。ＯＳ層２１９の通信制御部２１３が外部デバイスが用いる通信プロトコルに対応していればその機能を使えば良く、通信制御部２１３が外部デバイスが用いる通信プロトコルに対応していなければ外部デバイス通信部２２１がその通信プロトコルに従って通信する。

【００５０】

ＯＳ層２１９のインタプリタ２１４は、スクリプト層２１７で生成されたウェブ標準言語で記述された命令を解釈・実行する。例えば、画像の描画等の命令は、インタプリタ２１４を通して実行され、ディスプレイ１０４に表示される。

【００５１】

画像データ群２１５は、画像データを保持している領域である。データ保存部２２０は、必要に応じて、データ保持部２０４が保持する画像データを画像データ群２１５に保存させるために機能する。

【００５２】

レンダリング部２１６は、コンテンツ描画部２０５、画像処理制御部２０６、及びコンテンツ操作部２１０を制御して、処理対象の画像データのレンダリングを行う。このレンダリングには、例えば、スクリプト層２１７でプリンタ１１２への出力解像度で画像を生成することが含まれる。また、スクリプト層２１７におけるレンダリング結果、及び、スクリプト層２１７が生成途中の画像はディスプレイ１０４に表示されない。レンダリング結果は、ネイティブ層２１８のデータ変換部２０３に送信され、プリンタ１１２が利用できる形式の画像データに変換される。

【００５３】

＜ユーザ操作に伴う処理＞

図３はユーザ操作を含む処理を示すフローチャートである。まず、図３を用いて、Ｓ２１からＳ２８の各処理の概要を説明し、詳細は後述する。なお、本願のフローチャートの各ステップの処理は、情報処理装置１１５のＣＰＵ１００が、ＲＯＭ１０１あるいは２次記憶装置１０３に記憶されているプログラムを実行することにより実現される。また、図３に示す各ステップは、ＵＩの１つである図１２に示すアプリケーション画面１２００に対するユーザ操作に従って遷移する。アプリケーション画面１２００は、スクリプト層２１７によって生成される。アプリケーション画面１２００の操作は、例えば、タッチセンサ１０５を介して実現される。

【００５４】

Ｓ２１で、ＣＰＵ１００は、アプリケーション画面１２００の写真画像選択ボタン１２０１に対するユーザ操作（タッチ操作入力も含む。以後も同様）を検知すると、その操作に応じて、任意の画像を選択する。画像を選択すると、ＣＰＵ１００は、アプリケーション画面１２００の描画領域１２０６に選択された画像を表示する。

【００５５】

Ｓ２２では、ＣＰＵ１００は、表示されている画像の輝度を調整するためのスライドバー１２０２に対するユーザ操作を検知すると、そのユーザ操作に応じて、画像処理時に利用する補正パラメータを設定する。そして、ＣＰＵ１００は、設定した補正パラメータに

従って、表示されている画像に画像処理を施し、その処理内容及び処理結果を描画領域１２０６に表示する。

【００５６】

Ｓ２３は、ＣＰＵ１００は、スタンプ追加ボタン１２０３に対するユーザ操作を検知すると、スタンプ一覧を表示する。スタンプ一覧に対するユーザ操作によってスタンプの選択を検知すると、ＣＰＵ１００は、描画領域１２０６に選択されたスタンプを追加・表示する。

【００５７】

Ｓ２４で、ＣＰＵ１００は、アプリケーション画面１２００に対するユーザ操作に応じて、スタンプを特定する。スタンプの特定とは、ディスプレイ１０４にユーザ操作によってタッチされた座標とスタンプの座標より、スタンプがタッチされたか否かを判断するものである。スタンプがタッチされた場合、そのスタンプは操作受付状態となる。ここでは、ユーザ操作に応じて、スタンプが操作受付状態になっているものとする。操作受付状態については後述する。

【００５８】

Ｓ２５で、ＣＰＵ１００は、操作受付状態になっている表示領域内でスタンプをスワイプするユーザ操作を検知すると、そのユーザ操作に応じて、操作受付状態にあるスタンプが描画領域内を移動する。

【００５９】

Ｓ２６で、ＣＰＵ１００は、プリントボタン１２０５に対するユーザ操作を検知すると、プリントに必要な情報を設定するための設定画面１３０１（図１３）を表示する。プリントに必要な情報とは、例えば、図１３の設定画面１３０１に示されるように、用紙サイズ、用紙種類、印刷品位、縁あり／なしの設定項目がある。これ以外にも、両面／片面、モノクロ・カラー等、使用するプリンタが有する機能に応じて、設定可能な設定項目が構成される。

【００６０】

Ｓ２７で、ＣＰＵ１００は、設定画面１３０１の設定完了ボタン１３０２に対するユーザ操作を検知すると、描画領域１２０６に表示されている画像を、プリンタに出力するためのプリント解像度に変換するためのレンダリングを実行する。

【００６１】

Ｓ２８で、ＣＰＵ１００は、プリンタの解像度に変換された画像を、プリンタ制御のコマンドと共にプリンタ１１２に送信する。以上の処理により、ユーザにより選択された画像がプリンタ１１２でプリントされる。

【００６２】

尚、図３に示す処理は一例であり、処理内容はこれに限定されず、ステップ群の処理順序もこれに限定されるものではない。また、この実施形態において、プロセッサで翻訳され実行されるための命令セットを含む第１のプログラム層をスクリプト層２１７、プロセッサ以外で予め翻訳された命令セットを含む第２のプログラム層をネイティブ層２１８と定義する。そして、これらの第１のプログラム層と第２のプログラム層とを包含するプログラムがハイブリッドアプリケーションを実現する。文字列データ（テキストデータ）を第１の形式、バイナリデータを第２の形式と定義する。なお、スクリプト層２１７は、テキスト形式のデータを保持することが可能であり、ネイティブ層２１８は、バイナリ形式のデータを保持することが可能である。

【００６３】

＜写真画像選択とその画像処理の詳細＞

次に以上のような構成の情報処理装置において実行する写真画像選択とそれに伴う画像処理に関する幾つかの実施形態について説明する。従って、これらの場合、情報処理装置１１５は画像処理装置として機能することになる。

【００６４】

［実施形態１］

ユーザが図１２に示した写真画像選択ボタン１２０１を押下することでＳ２１が開始する。ここで、図３のＳ２１の写真画像選択の詳細について、図４を用いて説明する。なお、Ｓ３０１～Ｓ３０２、Ｓ３０９～Ｓ３１１はＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理であり、Ｓ３０３～Ｓ３０８はＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行する処理である。

【００６５】

Ｓ３０１で、ＣＰＵ１００は、一意なＩＤを生成する。このＩＤは、数値、文字列等、スクリプト層２１７からネイティブ層２１８へ送信できる形式であればどのような形でも良い。Ｓ３０２で、ＣＰＵ１００は、写真画像選択ボタン１２０１に対するユーザ操作に応じて画像選択をネイティブ層２１８に依頼する。Ｓ３０２では、Ｓ３０１で生成されたＩＤもネイティブ層２１８に送信される。その依頼では、バインディング機能によりスクリプト層２１７からネイティブ層２１８固有の画像選択ＡＰＩを直接呼び出す。しかしながら、直接ネイティブ層固有の画像選択ＡＰＩを呼び出せないのであれば、ネイティブ層２１８にスクリプト層２１７から直接呼び出せる関数を用意し、その関数内にネイティブ層固有の画像選択ＡＰＩを呼び出す関数を記述しておけばよい。これは、ラッパを予め用意しておく方法である。また、そのＡＰＩは、前記ＩＤを、例えば、引数として渡す仕組みを備える。このようにしてネイティブ層２１８へＩＤが渡される。

【００６６】

Ｓ３０３で、ＣＰＵ１００は、デバイス固有の画像選択ＵＩをディスプレイ１０４に表示する。表示された画像選択ＵＩに対するユーザ操作に基づいて、任意の画像を１枚選択する。なお、この実施形態ではデバイス内の画像フォルダから画像を１枚選択しているが、本発明はそれに限定されるものではない。例えば、インターネット上の画像や、脱着可能な記憶媒体内の画像を選択しても良いし、デバイスのカメラ機能を利用しその場で画像を撮影し、これを選択しても良い。

【００６７】

Ｓ３０４で、ＣＰＵ１００は、選択された画像を取得する。例えば、選択した画像が画像ファイルの状態であれば、ＣＰＵ１００は、ファイルを開き、その内容を読み取る。

【００６８】

Ｓ３０５で、ＣＰＵ１００は、取得した画像をＲＧＢ空間に展開する。ここでは、ＲＧＢデータをネイティブ層２１８に保持しているが、これに限定されるものではない。例えば、ＪＰＥＧ（Ｊｏｉｎｔ　Ｐｈｏｔｏｇｒａｐｈｙ　Ｅｘｐｅｒｔ　Ｇｒｏｕｐ）、ＰＮＧ（Ｐｏｒｔａｂｌｅ　Ｎｅｔｗｏｒｋ　Ｇｒａｐｈｉｃｓ）、ＲＧＢＡ形式などで保持することもできる。ＲＧＢＡ形式とは、画像データのＲＧＢ（赤、緑、青）成分に、透明度としてＡを組み合わせたものである。

【００６９】

Ｓ３０６で、ＣＰＵ１００は、展開したＲＧＢ画像を、スクリプト層２１７から取得したＩＤと関連付けてデータ保持部２０４に保持する。関連付け方法は、例えば、ＩＤとＲＧＢ画像を有するオブジェクトを作成することで、ＩＤによりＲＧＢ画像の特定を可能にするという方法が考えられる。この関連付け方法は、これに限らず、ＩＤと選択された画像のアクセス先であるパスや、ＩＤとＲＧＢ展開に応じて実行される関数やクラス等も考えられる。

【００７０】

Ｓ３０７で、ＣＰＵ１００は、展開したＲＧＢ画像をスクリプト層２１７で利用可能な形式（サポート可能な形式）の画像データに変換する。この実施形態では、変換するデータ形式をＪＰＥＧ（Ｊｏｉｎｔ　Ｐｈｏｔｏｇｒａｐｈｙ　Ｅｘｐｅｒｔ　Ｇｒｏｕｐ）とする。ＲＧＢ画像データからＪＰＥＧデータへの変換は、ＯＳ層２１９に備えられたエンコーダを利用すればよい。

【００７１】

なお、ここでは、ディスプレイ１０４の表示サイズに対してあまりにも大きな画像であれば、縮小してからＪＰＥＧへ変換する。スクリプトは画像をＵＩの大きさ、ディスプレ

イの解像度に合わせて縮小処理を行うことになる。ＵＩに対して過多な画素数の画像をスクリプトで扱うと、ＣＰＵに処理負荷が掛かり、かつ、大容量のメモリが必要となるため無駄が生じる。従って、この実施形態では、その無駄を回避するために縮小処理を実行する。縮小処理の目安としては、例えば、ディスプレイの表示サイズやＵＩで画像を表示する箇所のサイズを上限にし、それを越える画素数を持つ画像に対して縮小処理を行うよう制御するようにすれば良い。さて、その画像に対しては後述のように画像処理を施す場合があるかもしれない。これについても、元の画像（ＲＧＢ展開した画像データ）に対して画像処理を施し、上記の縮小処理を実行後に縮小された画像をスクリプト層に引き渡す。

【００７２】

Ｓ３０８で、ＣＰＵ１００は、ＪＰＥＧ形式のデータをＢＡＳＥ６４データに変換し、スクリプト層２１７へ送信する。これは、スクリプト層２１７では、ＲＧＢ画像のデータ配列をそのまま利用できないため、ネイティブ層２１８において、スクリプト層２１７で解釈可能な形式に変換する必要があるためである。ＪａｖａＳｃｒｉｐｔ（登録商標）では、文字列しか扱えないため、この実施形態では、文字列としてデータを表現するＢＡＳＥ６４の形式を利用する。

【００７３】

ここで、ＢＡＳＥ６４とは、バイナリデータを文字列データとして扱うためのエンコード方式である。スクリプト層２１７ではネイティブ層２１８と同様、画像データを取得し、変更を加えることが可能である。しかし、ネイティブ層２１８はＲＧＢ配列のように、ＲＡＭ１０２（メモリ）の連続した領域にその値を保持するのに対し、スクリプト層２１７では、連続した領域にデータが保持されているとは限らない。このように、スクリプト層２１７とネイティブ層２１８ではデータ保持方法が異なるため、従来、スクリプト層２１７から画像データを連続データとして一括してネイティブ層２１８へ送信することができなかった。

【００７４】

しかしながら、文字列データはスクリプト層２１７でも連続したデータとして扱われるので、ＣＰＵ１００は、スクリプト層２１７からＢＡＳＥ６４形式でネイティブ層２１８へ連続したデータを送信する。この実施形態では、ネイティブ層２１８は画像情報の変換処理を行う。即ち、スクリプト層２１７から受け取った文字列情報を画像データへ変換することで、ネイティブ層２１８でのデータ操作を可能としている。逆に、ネイティブ層２１８からスクリプト層２１７へ画像データを送信する際は、ネイティブ層２１８で扱う画像データをＢＡＳＥ６４方式を用いて文字列情報へと変換する。これにより、スクリプト層２１７とネイティブ層２１８との間で画像データの受け渡しが可能となる。

【００７５】

Ｓ３０９で、ＣＰＵ１００は、ネイティブ層２１８で変換されたＢＡＳＥ６４データを受信するとともに、そのＢＡＳＥ６４データを表示するための描画領域をＲＡＭ１０２に確保する。この実施形態では、描画領域の確保の一例としてＨＴＭＬのＣａｎｖａｓ機能を利用し、画像の描画は、Ｃａｎｖａｓの有するＣｏｎｔｅｘｔオブジェクトのＡＰＩを利用する。

【００７６】

Ｓ３１０で、ＣＰＵ１００は、補正パラメータを生成し、初期化する。ここで、補正パラメータとは、Ｓ２２の画像処理の内容を決定するパラメータ群を保持するオブジェクトである。ネイティブ層でどのような画像処理が施されるかは、補正パラメータによって決定される。ＪａｖａＳｃｒｉｐｔ（登録商標）で保持する補正パラメータの一例として、下記のような形態が考えられる。

【００７７】

```
var CorrectionParam = function(){
    this.brightness = 10;
}
```

この補正パラメータは、ＣｏｒｒｅｃｔｉｏｎＰａｒａｍオブジェクトの中に、明るさ

補正用にｂｒｉｇｈｔｎｅｓｓという名前の変数が設けられており、１０という値を格納するということを表している。

【００７８】

この実施形態では、説明の簡略化のために、補正パラメータは明るさ（輝度）補正用のみであるが、その他の補正処理用のパラメータ（ぼかしフィルタの強度、セピア変換のオン・オフ等）を追加し処理の種類を増やしても良いことは言うまでもない。

【００７９】

Ｓ３１１で、ＣＰＵ１００は、描画領域で描画するデータにＢＡＳＥ６４データを指定し、それに従って、その描画領域に画像を描画する。具体的には、ＣＰＵ１００は、スクリプト層で指定されたＢＡＳＥ６４データをＯＳ層のインタプリタ２１４に送信する。そして、インタプリタ２１４がＢＡＳＥ６４データのスクリプトを解釈し、描画領域に画像として表示する。ここで、描画領域に、ＢＡＳＥ６４データを反映させるサンプルコードの一例を示す。

【００８０】

```
-------------------------------------------------
var base64Data = ネイティブ層からのＢＡＳＥ６４データ
var canvas = document.createElement("canvas");  //画像の描画領域確保
canvas.setAttribute("width", 100);              //描画領域の大きさ設定
canvas.setAttribute("height", 100);
var context = canvas.getContext("2d");  //描画領域に描画するＡＰＩを持つオブジェクトの生成
var img = new Image();    //Imageオブジェクトの生成
img.src = base64Data;     //画像のＵＲＩを受け取ったＢＡＳＥ６４データとする
img.onload = function(){  //画像のロードが終わってから処理を開始する
  context.drawImage(img, 0, 0, img.width, img.height, 0, 0, canvas.width,
  canvas.height);  //contextオブジェクトのメソッドを用いて画像を描画領域に描画
  document.getElementById("div").appendChild(canvas);
  //本フローチャートではＣａｎｖａｓが何層にもなるレイヤー構造を想定している}
```

これらのＣａｎｖａｓは至る所好き勝手に存在するわけではなく、描画、移動、拡大等の操作は、特定の領域内（図１２の描画領域１２０６）で完結するその領域を指定しているものが「ｄｉｖ」であり、Ｃａｎｖａｓはそのｄｉｖに追加されてゆく形態をとる。

【００８１】

```
-------------------------------------------------
```

この実施形態では、描画領域１２０６に対しＣａｎｖａｓが追加されてゆくレイヤー構造を用いている。Ｃａｎｖａｓは一枚の画像として扱われるため、Ｃａｎｖａｓに写真画像を描画後、スタンプなどを追加すると、スタンプも合わせて一枚の画像となる。この場合、画像内のスタンプのみに対して特定・操作を行うことは難しい。これに対して、レイヤー構造を用いてＣａｎｖａｓを重ねて表示すれば、ユーザには一枚の絵に見えるが、実際の描画物はそれぞれ独立しており、各々に対し個別の操作を指示することができる。なお、スタンプは、画像データに対して付加されるので付加画像と呼ぶこともある。

【００８２】

こうして、画像とスタンプが一枚の絵（画像）としてＵＩ上に表示される。この実施形態では、その画像が印刷コンテンツとなる。ただし、この時点での画像はＵＩ上に表示され、操作や確認を行うためのものである。後述するように、印刷するためには印刷に適した解像度への変換が必要となる。

【００８３】

＜画像処理の詳細＞

ユーザが図１２に示したスライドバー１２０２を変化させることでＳ２２が開始する。ここでは、図３のＳ２２の画像処理の詳細について、図５を用いて説明する。なお、Ｓ４０１～Ｓ４０３、Ｓ４０９、Ｓ４１１は、ＣＰＵ１００がスクリプト層２１７のプログラ

ムを用いて実行する処理であり、Ｓ４０４～Ｓ４０８、Ｓ４１０はＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行する処理である。

【００８４】

Ｓ４０１で、ＣＰＵ１００は、補正パラメータを設定する。ここでは、図３のＳ３１０で生成した補正パラメータのｂｒｉｇｈｔｎｅｓｓの値を、スライドバー１２０２に対するユーザ操作に応じて設定される値が設定される。Ｓ４０２で、ＣＰＵ１００は、インジケータを起動し、ディスプレイ１０４に表示する。ここで、インジケータとは、ユーザに処理中である旨を伝える表示であり、一般には、プログレスバーや、時計マーク、図形の点滅や回転等の画像で表現する。Ｓ４０３で、ＣＰＵ１００は、Ｓ４０１にて設定された補正パラメータをネイティブ層２１８で利用できるＪＳＯＮ文字列の形式に変換する。ここで、補正パラメータはオブジェクトの形態を取っており、そのままではネイティブ層２１８で利用できないため、設定した補正パラメータをＪＳＯＮ文字列に変換する。そして、ＣＰＵ１００は、ＪＳＯＮ文字列に変換された補正パラメータを、図４のＳ３０１で生成したＩＤと共にネイティブ層２１８へ転送する。

【００８５】

Ｓ４０４で、ＣＰＵ１００は、ＪＳＯＮ文字列に変換された補正パラメータをデコードし、補正パラメータを取得する。より具体的には、補正パラメータをＯＳ層２１９に備えられているパーサを利用してパース（解析）する。パース後に、上記の例の場合、補正パラメータ内のｂｒｉｇｈｔｎｅｓｓが取得されることになる。

【００８６】

Ｓ４０５で、ＣＰＵ１００は、スクリプト層２１７から取得したＩＤを基に、図４のＳ３０５で展開されたＲＧＢ画像を特定する。尚、上述のように、ＩＤと画像の対応付けは、ＩＤとＲＧＢ画像と対にすることに限定されるわけではなく、例えば、ＩＤと画像のパスを関連付ける方法を用いてもよい。その他にＩＤと関連付ける例として、ネイティブ層２１８のオブジェクト、画像データの先頭アドレス、画像を呼び出す関数等、様々なものが考えられる。

【００８７】

Ｓ４０６で、ＣＰＵ１００は、取得した補正パラメータから施すべき画像処理を判断し、ＲＧＢ画像に対して画像処理を行う。この実施形態では、明るさ補正のパラ―メータより、全ての画素のＲＧＢの値に１０が加算される。前述の通り、補正パラメータに他の情報を追加し、画像処理の種類を増やしても良い。例えば、公知のモノクロ変換、公知のセピア色変換、「ＩｍａｇｅＦｉｘ」、「ＲｅｄｅｙｅＦｉｘ」、「ＳｍａｒｔＳｋｉｎ」などを追加することも可能である。

【００８８】

ここで、「ＩｍａｇｅＦｉｘ」とは、写真画像を、人物顔検出やシーン解析部を用いて自動で解析し、適切な明るさ・ホワイトバランス調整を行う機能（顔検出機能）である（特開２０１０－２７８７０８号公報参照）。また、「ＲｅｄｅｙｅＦｉｘ」とは、画像中から自動で赤目画像を検出して補正する機能（赤目検出機能）である（特開２００６－３５０５５７号公報参照）。また、「ＳｍａｒｔＳｋｉｎ」とは、写真画像から人物の顔を検出して、該顔の肌領域を好適に加工する機能である（特開２０１０－０１０９３８号公報参照）。なお、画像処理機能の種類は、これに限定されるものではなく、用途や目的に応じて、様々な画像処理を利用することができる。さらに、画像処理はＯＳ層２１９が提供する機能を利用してもよく、その場合であっても処理構成はこの実施形態と同じである。

【００８９】

Ｓ４０７で、ＣＰＵ１００は、画像処理が施されたＲＧＢ画像をスクリプト層２１７で利用可能な形式（サポート可能な形式）の画像データに変換する。ここでは、図３のＳ３０７と同様に、ＪＰＥＧ形式のデータに変換する。尚、前述のようにＵＩに対して画像サイズが大きい場合は、ＪＰＥＧデータに変換する前に縮小処理を実行する。Ｓ４０８で、ＣＰＵ１００は、スクリプト層２１７にインジケータの停止を依頼する。これは、ネイテ

ィブ層２１８から、スクリプト層２１７で定義されているインジケータ停止の関数を呼び出すことで実現する。

【００９０】

Ｓ４０９で、ＣＰＵ１００は、インジケータを停止して、ディスプレイ１０４の表示からインジケータを消去する。

【００９１】

一方、Ｓ４１０で、ＣＰＵ１００は、変換されたＪＰＥＧ形式のデータをＢＡＳＥ６４データに変換し、スクリプト層２１７へ送信する。

【００９２】

Ｓ４１１で、ＣＰＵ１００は、ネイティブ層２１８で変換されたＢＡＳＥ６４データを受信し、それに従って、図４のＳ３０９で確保した描画領域に画像を描画する。具体的には、ＣＰＵ１００は、スクリプト層で指定されたＢＡＳＥ６４データをＯＳ層のインタプリタ２１４に送信する。そして、インタプリタ２１４がＢＡＳＥ６４データのスクリプトを解釈し、指定したＵＩの描画領域に画像データの描画結果を表示する。以上の処理により補正パラメータに基づく画像処理が適用された画像データが表示される。

【００９３】

この実施形態では、図１２に示すようなスライドバー１２０２の変化によって画像処理が開始されるが、その開始方法はこれに限定されるものではない。例えば、画面にプラスボタン、マイナスボタンが配置されており、そのボタンを押すごとに明るさが調整されるという形態であっても良い。他にも、画像の右半分がタッチされたら明るさを増す、左半分がタッチされたら暗くするなど、タッチイベントと連動させた処理によって実現しても良い。また、ユーザ操作で補正パラメータだけ変化させておいて、画像処理の実行指示があった時に、全ての画像処理が一括して行われるという方法などを採用しても良い。

【００９４】

＜スタンプ追加の詳細＞

ユーザが、図１２に示すスタンプ追加ボタン１２０３を押下し、ハートスタンプ１２０８を選択すると、Ｓ２３の処理が開始する。ここでは、図３のＳ２３のスタンプ追加の詳細について、図６を用いて説明する。以下の説明では、ユーザ操作によって、図１２のアプリケーション画面１２００のスタンプ追加ボタン１２０３が押下されてスタンプ一覧が表示された後、ハートスタンプ１２０８が選択された場合を例に挙げる。なお、Ｓ５０１～Ｓ５０２、Ｓ５０８～Ｓ５１０は、ＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理であり、Ｓ５０３～Ｓ５０７はＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行する処理である。

【００９５】

Ｓ５０１で、ＣＰＵ１００は、一意なＩＤを生成する。このＩＤは、図３のＳ３０１で生成するＩＤと同等であるため、Ｓ３０１で説明した方法に従って生成される。Ｓ５０２で、ＣＰＵ１００は、Ｓ５０１で生成されたＩＤとともに、スタンプとして利用される画像のアクセス先（絶対パス）をネイティブ層２１８に送信することで、スタンプに対応するスタンプ画像の画像選択を依頼する。

【００９６】

Ｓ５０３で、ＣＰＵ１００は、スクリプト層２１７から受信したスタンプ画像の絶対パスとデバイス固有の画像選択ＡＰＩを利用して、スタンプ画像を取得する。Ｓ５０４で、ＣＰＵ１００は、取得したスタンプ画像をＲＧＢ展開する。Ｓ５０５で、ＣＰＵ１００は、展開したＲＧＢ画像を、スクリプト層２１７から取得したＩＤと関連付けてデータ保持部２０４に保持する。その関連付け方法は、図３のＳ３０６と同様である。Ｓ５０６で、ＣＰＵ１００は、ネイティブ層２１８で、展開したＲＧＢ画像をスクリプト層２１７で利用可能な形式（サポート可能な形式）の画像データに変換する。ここでの変換は、図３のＳ３０７と同様に、ＪＰＥＧ形式のデータに変換する。Ｓ５０７で、ＣＰＵ１００は、ＪＰＥＧ形式のデータをＢＡＳＥ６４データに変換し、スクリプト層２１７へ送信する。

【００９７】

Ｓ５０８で、ＣＰＵ１００は、ネイティブ層２１８で変換されたＢＡＳＥ６４データを受信するとともに、そのＢＡＳＥ６４データを表示するための描画領域をＲＡＭ１０２に確保する。Ｓ５０９で、ＣＰＵ１００は、オブジェクトパラメータを生成し、初期化する。ここで、オブジェクトパラメータとは、図３のＳ２７のレンダリング（詳細は後述）の際、レンダリング後のスタンプの回転角度を決定するために用いられるパラメータを保持するオブジェクトである。ＪａｖａＳｃｒｉｐｔ（登録商標）で保持するオブジェクトパラメータの一例としては、下記のような形態が考えられる。

【００９８】

```
var ObjectParam = function(){
        this.theta = 0;
        this.posX = 0;
        this.posY = 0;
        this.width = 100;
        this.height = 100;
}
```

このオブジェクトパラメータは、ＯｂｊｅｃｔＰａｒａｍオブジェクトの中に、回転角度を示すｔｈｅｔａという変数名と、ｔｈｅｔａには0という値が格納されているということを表している。同様に、ｐｏｓＸは描画領域（図１２の１２０６）の左上を基準点（図１２の１２１０）とした時のｘ座標、ｐｏｓＹは描画領域の左上を基準点とした時のｙ座標を表わしている。また、ｗｉｄｔｈは描画領域（図１２の１２０９）の横幅、ｈｅｉｇｈｔは描画領域の縦幅を表している。尚、この実施形態では、説明を簡単にするため、オブジェクトパラメータは必要最小限であるが、その他のパラメータ（平行移動量、拡大倍率等）を追加し、描画時やレンダリング時に利用できることは明らかである。

【００９９】

Ｓ５１０で、ＣＰＵ１００は、ＢＡＳＥ６４データを、生成したオブジェクトパラメータを基に、描画領域１２０６に画像として表示する。具体的には、ＣＰＵ１００は、選択されたスタンプに対応するＢＡＳＥ６４データをＯＳ層のインタプリタ２１４に送信する。そして、インタプリタ２１４がＢＡＳＥ６４データのスクリプトを解釈し、描画領域にスタンプ画像として表示する。

【０１００】

尚、この実施形態では、説明を簡単にするために、スタンプを１つ選択した場合を例に挙げているが、複数個のスタンプを選択できることは言うまでもない。また、この実施形態では、スタンプに予め用意した画像を利用しているが、Ｃｏｎｔｅｘｔオブジェクトを利用して描画物をスクリプト層で生成する方法を用いても良い。

【０１０１】

＜スタンプ特定の詳細＞

ユーザが図１に示したディスプレイ１０４にタップすることでＳ２４が開始する。「タップ」とは携帯型情報端末上で画面を指で押す操作のことをいい、これは、ＰＣでのポインティングデバイスによる「クリック」に相当する操作である。ここでは、図３のＳ２４のスタンプ特定の詳細について、図７を用いて説明する。なお、Ｓ６０２～Ｓ６０３は、ＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理であり、Ｓ６０１はＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行する処理である。

【０１０２】

Ｓ６０１で、ＣＰＵ１００は、ディスプレイ１０４上でタップされた座標を取得し、スクリプト層２１７に送信する。

【０１０３】

Ｓ６０２で、ＣＰＵ１００は、ネイティブ層２１８から受信した座標と、図５のＳ５０９で生成したオブジェクトパラメータの情報より、図３のＳ２３で追加したスタンプがタッチされたかどうかを判断する。追加したスタンプにおいて、オブジェクトパラメータは初期値のままである。そのため、上記のオブジェクトパラメータの一例に従えば、スタン

プは、描画領域１２０６の基準点１２１０を（０，０）とし、そこからｘ方向に１００、ｙ方向に１００の領域に描画されていることになる。これより、受信した座標の座標（ｘ，ｙ）の内、ｘ座標から描画領域１２０６のｘ座標分を差し引いた値が０～１００の範囲、かつ、ｙ座標から描画領域１２０６のｙ座標分を差し引いた値が０～１００の範囲であれば、スタンプはタッチされたと判断できる。スタンプが複数個ある場合は、より上層に表示されているスタンプから順番に判定してゆき、スタンプの特定がなされた時点で処理を終了すればよい。スタンプがタッチされたと判断された場合、そのスタンプは、図３のＳ２５の操作受付状態となる。

【０１０４】

以降の説明では、図３のＳ２４において、図３のＳ２３で追加したスタンプがタッチされたものとして説明する。

【０１０５】

Ｓ６０３で、ＣＰＵ１００は、判断結果に応じて、スタンプを操作受付状態に設定する。ここで、操作受付状態に設定することは、スタンプ操作に対する指示がなされたとき、スタンプがそれに応じた操作が実施される状態をいう。操作受付状態のスタンプが存在しない場合は、スタンプ操作に関する指示を行っても、何の変化も生じない。この実施形態では、スタンプの回転を行うスライドバー１２０４のみに言及しているが、他のスタンプ操作の構成を利用してもよいことは言うまでもない。

【０１０６】

＜スタンプ操作の詳細＞

スタンプが操作受付状態になった時点で図３のＳ２５が開始する。以下、図３のＳ２５のスタンプ操作の詳細について、図８を用いて説明する。なお、図８の各ステップは、ＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理である。

【０１０７】

Ｓ７０１で、ＣＰＵ１００はスタンプが「スワイプ」されたかどうかを、タッチイベントから取得できる情報に基いて調べる。ここで、「スワイプ」されたと判断された場合、処理はＳ７０２へ進み、そうでない場合はＳ７０４へ進む。ここで、「スワイプ」とは、携帯型情報端末をタップしたのち、指を画面から離さない状態で移動させる操作のことをいい、これはＰＣのポンティングデバイスを用いた「ドラッグ」操作に対応する。

【０１０８】

Ｓ７０２で、ＣＰＵ１００は、スタンプのオブジェクトパラメータのｐｏｓＸとｐｏｓＹの値を「スワイプ」によって得られたｘ座標とｙ座標に更新する。Ｓ７０３で、ＣＰＵ１００は、オブジェクトパラメータを用いて、更新されたｐｏｓＸ、ｐｏｓＹの値を基にして、描画領域１２０６内のスタンプ１２０９を更新された座標位置（ｐｏｓＸ，ｐｏｓＹ）へと移動させる。

【０１０９】

Ｓ７０４では、スタンプ操作後、画面のタッチが維持されているかどうかをタッチイベントから取得する。ここで、タッチが解除されていると判断された場合、スタンプ操作は終了し、図３のＳ２５の処理は終了する。これに対して、引き続きユーザによりタッチがされていると判断された場合、処理はＳ７０１に戻り、前述の処理を繰り返す。

【０１１０】

尚、この実施形態では、スタンプの操作はスワイプイベントによる移動のみであるが、本発明はこれによって限定されるものではない。例えば、ピンチアウトによるスタンプの拡大、ロングタップ（長押し）とスワイプを組み合わせてＣａｎｖａｓの描画物を平行移動させるなど、タッチイベントとコンテンツ操作には多くの組み合わせが考えられる。

【０１１１】

ここで、ピンチアウトとは、ユーザが二本の指で画面にタッチした後、二本の指の間隔を空ける操作である。一般的な携帯型情報端末において、この操作は、画像・画面の拡大などに利用される事が多い。また、先述の移動とは、描画領域１２０６内の移動とは異なり、描画物であるスタンプ１２０９がＣａｎｖａｓ内で移動することを意味している。例

えば、図１２において、スタンプ１２０９内のハートをｗｉｄｔｈ／２だけ右に移動したとすると、スタンプ１２０９の左半分は何も表示されず、その右半分にはハートの左半分が入っているような状態になる。

【０１１２】

＜プリンタ設定の詳細＞

ユーザが、図１２に示したプリントボタン１２０５を押下することで図３のＳ２６の処理が開始する。ここでは、図３のＳ２６のプリンタ設定の詳細について、図９を用いて説明する。なおＳ９０１、Ｓ９０９～Ｓ９１１はＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理であり、Ｓ９０２、Ｓ９０４～Ｓ９０５、Ｓ９０７、Ｓ９０８、Ｓ９１２はＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行する処理である。また、Ｓ９０３とＳ９０６とはプリンタ１１２が実行する処理である。

【０１１３】

Ｓ９０１で、ＣＰＵ１００は、スクリプト層２１７からネイティブ層２１８へデバイス情報としてのプリンタ情報取得を依頼する。これは、プリンタ１１２と通信を行うためのスクリプト層２１７からの要求にあたる。依頼の方法は、画像選択時と同様に、バインディング機能によりスクリプトからネイティブ固有のＡＰＩを呼び出す。ネイティブ層２１８には、スクリプト層２１７から直接呼び出せる関数もしくはその関数を間接的に呼び出す、いわゆる、ラッパが予め用意されている。例えば、ＧｅｔＰｒｉｎｔｅｒＩｎｆｏというネイティブ関数を用意しておき、スクリプト側からその関数を呼び出す。このようにして、ネイティブ層はスクリプト層からの外部デバイスとの通信の要求を取得する。

【０１１４】

通常、スクリプト層２１７からはセキュリティ上の制限で外部デバイスと直接通信することはできない。そのため、上記のように、スクリプト層２１７から、一旦、ネイティブ層２１８へ外部デバイス情報の取得を依頼し、ネイティブ層２１８を介して外部デバイスと通信を行う。ここで、ネイティブ層２１８は、ＯＳ層２１９を介して、外部デバイス（例えば、プリンタ１１２）と通信する機能を備えている。

【０１１５】

Ｓ９０２で、ＣＰＵ１００は、ネイティブ層２１８が該関数を呼び出されると、プリンタの検出、いわゆるディスカバリーを行う。例えば、同一無線ＬＡＮルータで繋がっているプリンタを検出する。ここでは、通信可能なプリンタの検出を行うため、例えば、Ｂｏｎｊｏｕｒ（登録商標）などのプロトコルにより、ブロードキャストやマルチキャスト等の方法を用いてプリンタに対して通信を試みる。

【０１１６】

これに対して、Ｓ９０３では、プリンタ１１２はその要求に応じて応答する。

【０１１７】

Ｓ９０４では、ＣＰＵ１００は、応答のあったプリンタのＩＰアドレスを検知して記憶する。さらに、Ｓ９０５では、ＣＰＵ１００は、応答のあったプリンタのＩＰアドレスへプリンタ情報の提供を要求する。応答のあったプリンタが複数の場合、ＣＰＵ１００は、全てのプリンタに対して情報の提供を要求する。そのために、ＣＰＵ１００はネイティブ層で、プリンタの情報を取得するためのコマンドを生成する。そのコマンドとは、プリンタの動作を指定する命令であり、その一例としては、以下のようなＸＭＬで表現される。

【０１１８】

```
01:     <?xml version="1.0" encoding="utf-8" ?>
02:     <cmd xmlns:trans="http://www.xxxx/yyyyy/">
03:         <contents>
04:             <operation>GetInformation</operation>
05:         </contents>
06:     </cmd>
```

上記各行の左側に書かれている「０１：」等の数値は説明を行うために付加した行番号であり、本来のＸＭＬ形式のテキストには記載されない。

【０１１９】

１行目はヘッダであり、ＸＭＬ形式で記述していることを表している。

【０１２０】

２行目のｃｍｄはコマンドの開始を意味する。ｘｍｌｎｓで名前空間を指定し、コマンドの解釈の定義を指定している。尚、６行目の＜／ｃｍｄ＞でコマンドの終了を示している。

【０１２１】

３行目は以後に内容を記載する宣言であり、５行目でその終了を示している。

【０１２２】

４行目には要求する命令が記載されており、＜ｏｐｅｒａｔｉｏｎ＞と＜／ｏｐｅｒａｔｉｏｎ＞の間に実際の命令文言が存在する。命令文言であるＧｅｔＩｎｆｏｒｍａｔｉｏｎは、外部デバイスであるプリンタの情報を取得する命令である。例えば、プリンタが対応している用紙種類、サイズ、縁なし印刷機能の有無、印刷品位、等のケーパビリティ情報を提供するよう要求する。

【０１２３】

尚、プリンタ情報取得コマンドの生成は、例えば、ＲＯＭ１０１に予め記憶した固定のテキストを読み込んでも構わない。また、ＸＭＬ等のテキスト形式に限らず、バイナリ形式で記述し、それに沿ったプロトコルで通信しても良い。また、生成したプリンタ情報取得コマンドは、ＨＴＴＰ等のプリンタが対応している通信プロトコルに従った形式で、外部デバイス通信部２２１を介してプリンタ１１２へ送信される。

【０１２４】

また、通信方法はこれに限定されるものではない。Ｗｉ－Ｆｉ（登録商標）ダイレクトやＢｌｕｅｔｏｏｔｈ（登録商標）、赤外線通信、電話回線、有線ＬＡＮ、ＵＳＢを用いた接続でも良く、その方法に沿ったプロトコルで通信を行えば同様の効果を得ることができる。

【０１２５】

図９では、ネイティブ層２１８で、コマンドを生成する構成としているが、スクリプト層２１７で、コマンドを生成する構成でも、同様の効果が得られる。その場合、スクリプト層２１７で、例えば、上記のＸＭＬ形式の命令文を含むコマンドを作成し、ネイティブ層２１８へ渡す。それを受けて、ネイティブ層２１８は、通信プロトコルに従った形式でプリンタ１１２へそのコマンドを送信する。

【０１２６】

Ｓ９０６で、プリンタ１１２は、情報処理装置１１５からコマンドを受信すると、デバイス情報であるプリンタ情報をＸＭＬ形式で通信プロトコルに沿って、情報処理装置１１５へ送信する。以下に、プリンタの情報の一例を示す。

【０１２７】

```
-----------------------------------------------
01:      <?xml version="1.0" encoding="utf-8" ?>
02:      <cmd xmlns:trans="http://www.xxxx/yyyyy/">
03:           <contents>
04:                <device id=”Printer001” />
05:                <memory receive= 7680000 />
06:                <mode = 1>
07:                     <media>GlossyPaper</media>
08:                     <size>A4</size>
09:                     <quality>1</quality>
10:                     <border>no</border>
11:                     <dpi x=1200 y=1200 />
```

```
12:                    </mode>
13:                    <mode = 2>
                              ～中略～
                       </mode>
                       <mode = 3>
                              ～中略～
                       </mode>

                       ～中略～

                  </contents>
             </cmd>
```

１行目はヘッダであり、ＸＭＬ形式で記述していることを表している。２行目のｃｍｄはコマンドの開始を意味する。ｘｍｌｎｓで名前空間を指定し、コマンドの解釈の定義を指定している。尚、最下行の＜／ｃｍｄ＞でコマンドの終了を示している。

【０１２８】

３行目は以後に内容を記載する宣言であり、下の＜／ｃｏｎｔｅｎｔｓ＞までその内容は継続する。４行目でデバイスＩＤを示している。ここでは、プリンタ１１２の機種名が「Ｐｒｉｎｔｅｒ００１」であることを表している。５行目についてはこの実施形態では用いない。後の実施形態で詳細に述べる。６行目以降はプリンタ１１２が有する各モードについての記述である。＜ｍｏｄｅ＞から＜／ｍｏｄｅ＞までで、１つのモードにおける情報が記述されている。６行目では、モードの番号が１である。以降の＜ｍｅｄｉａ＞は印刷用紙の種類、＜ｓｉｚｅ＞は用紙サイズ、＜ｑｕａｌｉｔｙ＞は印刷品位、＜ｂｏｒｄｅｒ＞は縁あり／なしの情報をそれぞれ記述している。１１行目の＜ｄｐｉ＞は入力解像度を表しており、横方向が１２００［ｄｐｉ］、縦方向が１２００［ｄｐｉ］である。入力解像度についての詳細は後述する。

【０１２９】

１３行目以降は他のモードであるモード２についての情報が記述されている。このように、プリンタ１１２の機種名と、そのプリンタが対応している全てのモードがこのＸＭＬに記述されている。尚、プリンタ情報の記述方法はこれに限定されることはなく、タグ形式でないテキストや、バイナリ形式等の他の形式であっても構わない。

【０１３０】

また、上記はプリンタの印刷機能の情報を受け渡ししている例を記載しているが、本発明はこれに限定されるものではない。例えば、プリンタが処理可能な画像処理や解析処理、静かに印刷するモードの有無、メモリカードの利用有無、インク残量などのステータスなどの情報を受け渡ししても良い。画像処理の例としては、モノクロやセピア、彩度強調などの色変換、複数画像のレイアウト、ホワイトバランス補正、ノイズ除去、その他自動で写真を好ましい色や輝度に補正する処理などが挙げられる。

【０１３１】

そして、Ｓ９０７では、ＣＰＵ１００は、プリンタからプリンタ情報を取得する。ＣＰＵ１００は、ネイティブ層で、受信したプリンタ情報から、例えば、プリンタ１１２が有する全てのモードにおける印刷用紙の種類、サイズ、印刷品位、縁あり／なしの項目と項目数等を取得する。

【０１３２】

Ｓ９０８では、ＣＰＵ１００は、受信したプリンタ情報をスクリプト層２１７が解釈可能な形式に変換して、スクリプト層２１７へ送信する。つまり、ＣＰＵ１００は、プリンタ１１２との通信によって得られた情報をスクリプト層２１７へ渡す。具体的には、ネイティブ関数を設けておき、バインディング機能を用いる。また、受け取ったＸＭＬ形式のプリンタ情報で送信したり、タグなしのテキスト形式に変えて送信する等の方法を用いて

も良い。加えて、スクリプト層２１７から特定のネイティブ関数を呼び出す毎に、その戻り値として情報を取得しても良い。また、ネイティブ関数に取得するモードなどの引数を渡し、その戻り値として情報を得ても良い。さらに、上述のＪＳＯＮ文字列を用いた受け渡しや、データ変換部２０７及び２０３を用いてＢＡＳＥ６４等の文字列で受け渡しを行ってもよい。

【０１３３】

Ｓ９０９で、ＣＰＵ１００は、ネイティブ層２１８から受信したプリンタ情報に基づいて、プリンタ１１２で利用できる機能を含む設定画面（図１３）を形成し、表示する。これは、表示制御とも呼ばれる。ここで、接続可能なプリンタが複数ある場合はプリンタ名を表示し、設定画面１３０１を表示する前にユーザに印刷するプリンタを選択させるための表示画面を生成する。そして、ＣＰＵ１００は、選択されたプリンタに対応するプリンタ情報を用いて、選択されたプリンタの設定画面を表示する。尚、プリンタの選択は、上記に限らず、一番早く応答してきたプリンタや、より機能が多いプリンタ、印刷ジョブが混雑していないプリンタを選択する、等の方法も考えられる。

【０１３４】

このように、ＣＰＵ１００は、印刷用紙の種類・サイズ、印刷品位、縁あり／なし等のプリンタで利用できる機能を選択させる設定画面１３０１（図１３）を表示する。設定画面の形成方法の一例として、以下に、ＨＴＭＬ記述のサンプルを示す。

【０１３５】

```
-----------------------------------------------
<!DOCTYPE html>
    <head>
    <title>印刷設定 </title>
    <script>
        <!-- 用紙サイズ -->
        var PaperSizeNum = GetPaperSizeNum();
        var p = document.getElementById("PaperList");
        var i;
        for(i=0; i<PaperSizeNum; i++){
              p.options[i] = new Option(GetPaperSizeT(i), GetPaperSizeV(i));
        }

        <!-- 用紙種類-->
        var MediaTypeNum = GetMediaTypeNum();
        var m = document.getElementById("MediaList");
        var j;
        for(j=0; j<MediaTypeNum; j++){
              m.options[i] = new Option(GetMediaTypeT(j), GetMediaTypeV(j));
        }

        <!-- 印刷品位 -->
        var QualityNum = GetQualityNum();
        var q = document.getElementById("QualityList");
        var k;
        for(k=0; k< QualityNum; k++){
              q.options[i] = new Option(GetQualityT(k), GetQualityV(k));
        }

        <!-- 縁あり/なし-->
        var BorderNum = GetBorderNum();
```

```
        var b = document.getElementById("BorderList");
        var l;
        for(l=0; l<BorderNum; l++){
              b.options[i] = new Option(GetBorderT(l), GetBorderV(l));
        }

        <!-- 印刷関数-->
        function printer() {
              SetPrint(document.getElementById("PaperList").value,
                    document.getElementById("MediaList").value,
                    document.getElementById("QualityList").value,
                    document.getElementById("BorderList").value);
        }
    </script>
    </head>

        <!-- 表示部 -->
        <body>
        用紙サイズ  <select id="PaperList"></select><br />
        用紙種類    <select id="MediaList"></select><br />
        印刷品位    <select id="QualityList"></select><br />
        縁あり/なし <select id="BorderList"></select><br />
        <br />

    <button id="btn1" onclick="printer()">設定完了</button>
          </body>
  </html>
```

上記のＧｅｔＰａｐｅｒＳｉｚｅＮｕｍ（　）、ＧｅｔＭｅｄｉａＴｙｐｅＮｕｍ（　）、ＧｅｔＱｕａｌｉｔｙＮｕｍ（　）、ＧｅｔＢｏｒｄｅｒＮｕｍ（　）はネイティブ関数であり、それぞれの項目数を取得する機能を備える。例えば、プリンタが対応している用紙サイズがＡ４、Ａ５、Ｂ５、Ｌ判の４種類である場合、ＧｅｔＰａｐｅｒＳｉｚｅＮｕｍ（　）は４を返す。

【０１３６】

ＧｅｔＰａｐｅｒＳｉｚｅＴ（ｎ）、ＧｅｔＭｅｄｉａＴｙｐｅＴ（ｎ）、ＧｅｔＱｕａｌｉｔｙＴ（ｎ）、ＧｅｔＢｏｒｄｅｒＴ（ｎ）もネイティブ関数であり、引数ｎの値番目の文字列を返す。例えば、用紙サイズのテキストを返す関数のＧｅｔＰａｐｅｒＳｉｚｅ（０）の返り値は「Ａ４」、ＧｅｔＰａｐｅｒＳｉｚｅ（１）の返り値は「Ａ５」となる。これらの値は、プリンタから受信するプリンタ情報からネイティブ関数が取り出す。

【０１３７】

ＧｅｔＰａｐｅｒＳｉｚｅＶ（ｎ）、ＧｅｔＭｅｄｉａＴｙｐｅＶ（ｎ）、ＧｅｔＱｕａｌｉｔｙＶ（ｎ）、ＧｅｔＢｏｒｄｅｒＶ（ｎ）もネイティブ関数であり、引数ｎの値番目の文字列を返す。例えば、用紙種類のテキストを返す関数のＧｅｔＭｅｄｉａＴｙｐｅＴ（ｎ）の返り値は「光沢紙」のように、表示してユーザに示す文言である。これに対して、ＧｅｔＭｅｄｉａＴｙｐｅＶ（０）の返り値は「ＧｌｏｓｓｙＰａｐｅｒ」とプリンタが解釈可能な表現となっている。

【０１３８】

これらの文言や表現は、プリンタから送られてきた情報と結び付けてネイティブが決定する。例えば、プリンタから送られてきた情報より取り出した値が「ＧｌｏｓｓｙＰａｐ

ｅｒ」であった場合、表示するテキストは「光沢紙」と決定する。決定の方法として、ネイティブはこれらの対応表を予め保持しておき、その対応表に沿ってテキストを決定すれば良い。

【０１３９】

尚、上記では、例として、用紙サイズ、用紙種類、印刷品位、縁あり／なしの設定を行う仕様であるが、これに限定されるものではない。他の例として、両面／片面、カラー／モノクロ、画像補正のオン／オフ等の他の設定項目が挙げられることは言うまでもない。また、前述のように印刷機能のみではなく、プリンタが処理可能な画像処理や解析処理、静かに印刷するモードの有無、メモリカードの利用有無、インク残量などのステータスなどの情報を表示しても良い。

【０１４０】

Ｓ９１０で、ＣＰＵ１００は、設定画面１３０１に対するユーザ操作に基づいて、プリンタに設定する機能を選択する。上記の例で示したＨＴＭＬを、レンダリング部２１６を用いてディスプレイ１０４に表示した例が図１３に示す１３０１である。ネイティブ層２１８を介してプリンタ情報を要求し、プリンタ情報から上記のネイティブ関数を用いて取得した情報を基に設定画面１３０１が形成されている。尚、上記ＨＴＭＬはスクリプト層２１７、ネイティブ層２１８のいずれで形成しても良い。

【０１４１】

また、図１３に示す用紙サイズ等の設定項目はそれぞれプルダウンメニューになっており、ユーザ操作によって項目を選択する事ができる。ここで、設定画面１３０１は、プルダウンメニューによって、用紙サイズの設定項目として選択可能な項目の一覧が表示されている状態を示しており、ユーザ操作によってＡ４やＡ５等の用紙サイズを選択することができる。

【０１４２】

Ｓ９１１で、ＣＰＵ１００は、設定完了ボタン１３０２に対するユーザ操作を検知すると、ユーザ操作によって選択された設定項目を含む設定情報を作成して、ネイティブ層２１８へ送信する。上記ＨＴＭＬ記述の例にあるＳｅｔＰｒｉｎｔ（　）もバインディング機能を有するネイティブ関数である。上記の例では、ＳｅｔＰｒｉｎｔ（　）を用いて、用紙サイズ、用紙種類、印刷品位、縁あり／なしの設定を文字列としてネイティブ層２１８へ渡している。

【０１４３】

Ｓ９１２で、ＣＰＵ１００は、バインディング機能によりスクリプト層２１７から設定情報を受信する。ネイティブ層２１８では、後に、受信した設定情報、印刷対象の画像データ、さらにはスタンプの画像データとを基に、プリンタ１１２の通信プロトコルに従って、プリントコマンドを生成する。そして、プリンタコマンドは、通信制御部２１３を介してプリンタ１１２へ送信されることになる。

【０１４４】

＜レンダリングの詳細＞

ユーザが、図１３に示した設定画面１３０１の設定完了ボタン１３０２を押下することで図３のＳ２７のレンダリング処理が開始する。ここでは、図３のＳ２７のレンダリングの詳細について、図１０を用いて説明する。

【０１４５】

この実施形態のレンダリングは、ＵＩに表示しているコンテンツをプリント用の画像サイズ（画素数）に変換する事である。コンテンツとは、上述した写真画像とスタンプを配置した画像である。通常、ＵＩに表示される画像の画素数よりも印刷で用いられる画像の画素数の方が多いため、印刷には画像の拡大処理が必要となる。ここでは、ＵＩに表示されたコンテンツを同じ構図でより大きな画像（画素数）になるように描画する事で、拡大処理を行う。この実施形態では、バンドレンダリング（分割拡大）の処理を主にスクリプト層で実行する。

【０１４６】

なお、図１０において、Ｓ１００１、Ｓ１００３、Ｓ１００４、Ｓ１００７は、ＣＰＵ１００がスクリプト層２１７のプログラムを用いて実行する処理である。Ｓ１００２、Ｓ１００５、Ｓ１００８は、ＣＰＵ１００がＯＳ層２１９のプログラムを用いて実行する処理である。Ｓ１００６は、ＣＰＵ１００がスクリプト層２１７とネイティブ層２１８のプログラムを用いて実行する処理である。

【０１４７】

Ｓ１００１で、ＣＰＵ１００は、インジケータの起動をＯＳ層２１９に依頼する。Ｓ１００２で、ＣＰＵ１００は、依頼によって起動したインジケータをディスプレイ１０４に表示する。これにより、ユーザは処理が実行中であることを知ることができる。

【０１４８】

Ｓ１００３で、ＣＰＵ１００は、Ｓ９１１で確定した設定情報で設定されている用紙サイズに対応する出力サイズを決定する。出力サイズとは画像の画素数の事であり、この実施形態で出力サイズは、上記の設定で指定された印刷モードでの入力解像度と用紙サイズより算出する。入力解像度とは、プリンタ１１２の印刷モード毎に定められており、その入力解像度の画素数で画像処理や量子化処理が行われる。その入力解像度以外の画像データがプリンタ１１２へ転送された場合、プリンタでその入力解像度になるように拡大・縮小（変倍）が行われる。

【０１４９】

ここでは、入力解像度が１２００ｄｐｉ、用紙サイズがＡ４であるとして出力サイズの算出方法の例を後述する。なお、「入力画素数［ｐｉｘｅｌ］」が出力サイズにあたり、入力解像度はプリンタで設定されている印刷モードによって異なる場合がある。印刷モードとは用紙種類や品位などによって設けられた設定である。他の例として、出力サイズは用紙サイズ毎に予め定めておいても良いし、予め定めた解像度から算出しても良い。例えば、入力解像度を３００ｄｐｉなどに設定しておき、用紙サイズを用いて算出しても良い。

【０１５０】

なお、プリンタの印刷モードでの入力解像度以下の画像データをプリンタへ転送して印刷する場合は、プリンタ内で印刷モードの入力解像度へと拡大処理が実行される。しかし、公知の単純なロジックを用いて拡大処理を実行する場合、印刷画像にスタンプなどのオブジェクトの線や写真画像にボケやジャギーが発生し、人の目には画像品質が劣化したように見えてしまう。これとは反対に、プリンタの印刷モードの入力解像度以上の画像データが転送されたとしても、その画像データはプリンタ内で入力解像度に縮小されてしまう。

【０１５１】

従って、上記のように出力サイズをプリンタの印刷モードでの入力解像度と用紙サイズより決定することが、処理効率が良く、結果として得られる画像の画質も良い。

【０１５２】

Ｓ１００４では、出力画像の描画領域のサイズを算出する。

【０１５３】

ここで、出力画像の描画領域とは一枚のＣａｎｖａｓであり、出力サイズのＣａｎｖａｓに対して、先述した個々のＣａｎｖａｓを、サイズと領域内の相対位置が合うようにレンダリングしてゆく。

【０１５４】

この実施形態において、出力画像の描画領域のサイズは、アプリケーションが動作している情報処理装置の状況により算出される。これは、その情報処理装置でのメモリ不足を回避するためである。例えば、入力解像度が１２００ｄｐｉのプリンタで用紙サイズがＡ４の印刷を行う場合、メモリに１．２５［ＧＢ］の容量が必要となる。そのメモリ容量の算出方法を以下に示す。

【０１５５】

サイズＡ４：縦２１０［ｍｍ］×横２９７［ｍｍ］
縦画素数［ｐｉｘｅｌ］
＝２１０［ｍｍ］÷２５．４[ｍｍ／ｉｎｃｈ]×１２００［ｄｐｉ］
横画素数［ｐｉｘｅｌ］
＝２９７［ｍｍ］÷２５．４[ｍｍ／ｉｎｃｈ]×１２００［ｄｐｉ］
入力画素数［ｐｉｘｅｌ]
＝縦画素数［ｐｉｘｅｌ］×横画素数［ｐｉｘｅｌ］
必要メモリ容量［Ｂｙｔｅ］
＝入力画素数［ｐｉｘｅｌ］×３［ｃｈａｎｎｅｌ］×８［ビット］
４１７．６［ＭＢ］（バイナリデータの場合）

ここで、３［ｃｈａｎｎｅｌとは、画像データの各画素がＲＧＢの３成分で表現されることを示している。画像データの各画素がＹＭＣＫの４成分で表現されるなら、４［ｃｈａｎｎｅｌ］となる。また、８［ビット］とは、各画素各色成分が０～２５５の値をとる８ビットで表現されることを示している。

【０１５６】

ここで、８ビットの値（０～２５５）をすべて３桁の文字列で扱うとすると、必要メモリ容量（文字データ）は４１７．６［ＭＢ］×３［文字］　１．２５［ＧＢ］となる。

【０１５７】

出力画像の描画領域のサイズの決定方法として、例えば、メモリ確保を試みて、確保できなければ確保する量を減らして再度試み、確保できるまでそれを続けるなどの方法がある。そのメモリ確保はＣａｎｖａｓのｓｅｔＡｔｔｒｉｂｕｔｅで横幅（ｘ方向）と高さ（ｙ方向）を設定し、ｇｅｔＣｏｎｔｅｘｔで領域確保を行うことでなされる。ここで、横幅はプリンタに出力する全体画像の横幅で固定し、縦幅（高さ）を減らしながら上記のメモリ確保の試みを行う。こうする事で、図１４に例示するように出力する全体画像を帯状（バンド）に分ける事が可能である。

【０１５８】

なお、この実施形態では横幅を固定し、高さ方向に帯状に分割する形にしたが、本発明はこれによって限定されるものではない。例えば、正方形や横方向に分割を行っても良い。ただし、プリンタでの印刷は画像の上部から下部に向かって順になされるのが一般的である。従って、プリンタに入力する画像データもこの印刷順序に対応して横に分割した帯状にすることが望ましい。このようにして、後述する分割拡大の処理が進む毎に、プリンタに画像データの分割送信が可能となり、印刷終了までの時間を短縮するという工夫も可能である。以上の処理から、帯状（バンド）に分割してレンダリング（プリンタの入力解像度に合わせて画像データを拡大処理して描画すること）を行うことをバンドレンダリングと呼ぶ。

【０１５９】

なお、出力画像の描画領域のサイズの決定方法は上記に限らず、例えば、確実にメモリ不足が生じないと予想される値に予め定めておいても良いし、レンダリングで印刷に遅延が起きない容量に決定するなどしても良い。

【０１６０】

この実施形態では上記のように出力画像の描画領域のサイズを決定する事により、スクリプト層におけるメモリ不足の発生を防いでいる。

【０１６１】

Ｓ１００５で、ＣＰＵ１００は、算出した出力画像の描画領域をＲＡＭ１０２に確保する。

【０１６２】

Ｓ１００６で、ＣＰＵ１００は、バンドレンダリングを行う。なお、実際のレンダリング処理は、スクリプト層からの指示を受けてＯＳ層２１９のインタプリタが実行するが、ここでは、スクリプトがＯＳの機能を利用する事もスクリプト層での実行として表してい

る。その際、写真画像がＳ１００５で確保した出力画像の描画領域に描画されるようにする。Ｓ１００６で、ＣＰＵ１００は、写真画像の描画の依頼のスクリプトを解釈し、写真画像をレンダリングする。画像の描画はＯＳ層２１９で行われ、ＣａｎｖａｓのＣｏｎｔｅｘｔオブジェクトのｄｒａｗＩｍａｇｅメソッドを利用する。なお、ここでいう描画とは、出力画像の描画領域として確保したメモリ空間へ画像データとして出力することをいい、ディスプレイ１０４上に画像を表示することではない。この実施形態ではＳ１００６でバンドレンダリングの手法を用い、分割して画像を描画する。以下、その詳細について説明する。

【０１６３】

図１４は描画領域に出力される画像とバンドレンダリングとの関係を模式的に示す図である。

【０１６４】

図１４において、１４０１はＵＩ上に表示されたＣａｎｖａｓの画像、１４０２、１４０４はユーザの指示により配置されたスタンプ、１４０３はユーザが選択した写真画像である。尚、拡大に用いる写真画像はＵＩ表示されているものではなく、画像ファイルをオープンして原画像を読取ったものを用いる。上述のように、ＵＩ用の表示には縮小処理が施されたものを使っている場合がある。縮小画像は原画像の情報を失っており、これを拡大しても失われた情報は元に戻らないからである。そのため、拡大対象の画像として、ＵＩ用の画像ではなく、ファイルから読み取られた画像サイズのものが使用される。尚、上述で補正等の画像処理を施した場合、画像ファイルをオープンして読み取った画像に対して該画像処理を施したものを拡大対象とする。ここでは、図４のＳ３０４で取得し、Ｓ３０５でＲＧＢ展開した画像をネイティブ層２１８から受け取り、スクリプト層２１７がその画像に拡大処理を行う。

【０１６５】

スタンプなどの画像はベクタデータとして保持しているものがある。ベクタデータは描画する際に画像として作成されるデータであるため、例えば、Ｃａｎｖａｓで拡大後に描画する場合にはその画質が劣化しない。ただし、一度ビットマップデータとして描画されたものやベクタデータでない画像データのスタンプは写真画像と同様である。この場合、上記の写真画像と同様、ＵＩ表示されているものではなく、なるべく解像度が高いソースを用いる。

【０１６６】

この実施形態ではＣａｎｖａｓの画像１４０１をプリンタ１１２が求める入力解像度（画素数）へ拡大する。拡大前のＣａｎｖａｓの画像１４０１は画像１４０５と同じサイズであり、横幅がＷｂ［ｐｉｘｅｌ］、高さ（縦長）Ｈｂ［ｐｉｘｅｌ］である。１４０６は拡大後の画像であり、横幅Ｗｅ［ｐｉｘｅｌ］、高さ（縦長）Ｈｅ［ｐｉｘｅｌ］である。また、１４０７、１４１０、１４１３は同サイズの出力画像の描画領域であり、横幅Ｗｅ［ｐｉｘｅｌ］、高さ（縦長）Ｈｐ［ｐｉｘｅｌ］である。

【０１６７】

上述のように拡大後の画像の画素数は多くなるため、メモリ不足が生じやすい。そのため、図１４に示す例では、３つに分割して拡大後の画像データを保持、あるいは外部デバイス（例えば、プリンタ）へ送信する。つまり、ここでは、拡大処理後の画像データの容量に対して１／３の容量であれば格納可能なメモリを出力画像の描画領域として確保できた例を説明する。なお、この実施形態では簡潔に説明するために、拡大画像を３つに分割しているが、本発明はこれによって限定されるものではなく、実際は実行環境のメモリの状態によって出力画像の描画領域と分割数を定めれば良い。

【０１６８】

次に、分割拡大の一例としてのバンドレンダリングの処理についてフローチャートを参照して説明する。

【０１６９】

図１５はバンドレンダリング処理の詳細を示す図である。なお、図１５において、Ｓ１

５０１～Ｓ１５０６、Ｓ１５０８はＣＰＵ１００がスクリプト層２１７のプログラムを実行する処理であり、Ｓ１５０７はＣＰＵ１００がネイティブ層２１８のプログラムを実行する処理である。

【０１７０】

まず、Ｓ１５０１では分割数を算出する。分割数は以下の式に従って決定する。即ち、

分割数 ＝Ｈｅ［ｐｉｘｅｌ］／Ｈｐ［ｐｉｘｅｌ］

である。ただし、小数点以下は切り上げる。ここで、ＨｐはＳ１００４で算出された出力画像領域の高さである。

【０１７１】

Ｓ１５０２では拡大率を決定する。

【０１７２】

拡大率はＵＩ表示されたＣａｎｖａｓのサイズ（画像１４０１、１４０５）に対する画像１４０６の比であり、以下のようになる。即ち、

ＳｃａｌｅＷ（横倍率）＝Ｗｅ／Ｗｂ

ＳｃａｌｅＨ（縦倍率）＝Ｈｅ／Ｈｂ

である。

【０１７３】

Ｓ１５０３では、変数ｉに“０”を代入し、分割数だけループ処理を実行する。

【０１７４】

次に、Ｓ１５０４では、画像１４０１に含まれるスタンプ１４０２、１４０４と写真画像１４０３の移動量を設定する。その移動は上方への平行移動であり、その移動距離はＨｂｐ×ｉである。ここで、初回はｉ＝０のため、移動は行われない。移動の詳細な説明は後述する。

【０１７５】

Ｓ１５０５では拡大率を設定する。この実施形態での拡大率の設定は、Ｃａｎｖａｓのｓｃａｌｅ（ ）メソッドを使用し、次のように設定する。

【０１７６】

ｓｃａｌｅ（ＳｃａｌｅＷ，ＳｃａｌｅＨ）；

そして、Ｓ１５０６で出力画像の描画領域の出力を行う。

【０１７７】

この実施形態ではＣａｎｖａｓのｄｒａｗＩｍａｇｅ（ ）メソッドにより、出力画像の描画領域へ画像データを出力する。その際、画像１４０６の全体は出力画像の描画領域１４０７には入りきらない。そのため、ｄｒａｗＩｍａｇｅ（ ）は画像１４０６の上端から出力画像の描画領域へ画像データを出力していき、出力画像の描画領域の下端へ到ると出力を終了する。よって、出力画像の描画領域へは画像１４０６の点線部から上部を切り出したような画像が描画領域１４０７に出力される。

【０１７８】

Ｓ１５０７で、出力画像の描画領域の画像データをファイルへ出力する。この場合、スクリプト層２１７はネイティブ層２１８を介してファイルへ出力する。この出力形態は、この実施形態ではＲＧＢビットマップ形式とする。そのファイルの保存場所は２次記憶装置１０３なので、容量に関してメモリ（ＲＡＭ１０２）への直接的な影響がない。そのため、メモリ（ＲＡＭ）の不足に関する問題は生じない。

【０１７９】

ここではバインディング機能によりスクリプト層からネイティブ層の関数を呼び出す。この際、保存する画像データの縦、横の画素数も引数として渡す。ネイティブ層は渡された縦×横の画素数分の画像データをファイルに保存する。前述のようにスクリプト層では文字列で数値を保持しているため、ネイティブ層にてバイナリデータに変換してからファイルへ保存を行う。

【０１８０】

なお、以上の例では２次記憶装置１０３へファイルとして画像データを保存したが、本

発明はこれによって限定されるものではない。例えば、メモリカードや外部の記憶装置、サーバ、インターネットを介したクラウドシステムへ保存しても良い。加えて、ＯＳの制約でスクリプト層２１７ではメモリに大きな容量を確保することはできないが、ネイティブ層２１８では大きな容量のメモリを扱えることがある。その際は、ネイティブ層２１８で管理するメモリ（ＲＡＭ）に分割拡大した画像データを保持しても良い。

【０１８１】

Ｓ１５０８では分割数分の処理を終えたかどうかを調べる。ここで、分割数分の処理を全て終えていないと判断された場合、ｉの値を“＋１”インクリメントして処理はＳ１５０４へ戻り、全て終えている場合はバンドレンダリングの処理を終了する。

【０１８２】

次に、３分割での出力中央部の拡大について説明する。ここは、２回目のループ処理（ｉ＝１）にあたる。

【０１８３】

Ｓ１５０４では、画像１４０１に含まれるスタンプ１４０２、１４０４と写真画像１４０３の移動量を設定する。その移動は上方への平行移動であり、その移動距離はＨｂｐ×ｉである。ここでは移動にＣａｎｖａｓのｔｒａｎｓｌａｔｅ（　）メソッドを用いる。ここで、Ｈｂｐは、画像１４０１を３分割したときの高さである。以下の式で表す事ができる。即ち、

Ｈｂｐ＝Ｈｐ×Ｈｂ／Ｈｅ

である。

【０１８４】

こうして平行移動の設定を行った際の仮想図が図１４の１４０８である。その太線枠はＣａｎｖａｓで出力される範囲であり、この太線枠の外は出力（又は描画）されない。ここでは、図１４の１４０８に示すように、画像１４０１を上方にＨｂｐだけ平行移動する設定を行う。

【０１８５】

そして、上述した上部の拡大と同様に、Ｓ１５０５ではＣａｎｖａｓのｓｃａｌｅ（）メソッドで拡大率を設定する。拡大の倍率は前述した上部と同じである。図１４の１４０９はその拡大率で拡大した際の仮想図である。図１４の１４０８のように上方にスライドし、ｓｃａｌｅ（　）メソッドで拡大設定した事により、拡大後の画像がＨｐだけ上へ平行移動している。なお、移動、拡大の順で設定を行ったが、拡大、移動の順であっても構わない。その際の移動量は、拡大後の画像に合わせた値（この例ではＨｐ）とする。

【０１８６】

続いて、Ｓ１５０６では上記と同様にＣａｎｖａｓのｄｒａｗＩｍａｇｅ（　）メソッドにより、出力画像の描画領域へ画像データを出力する。図１４の１４０６～１４０７を参照して上述したように、図１４の１４０９の太線枠の点線部より上部のみが出力画像の描画領域へ出力される。その出力の結果は図１４の１４１０として示されており、これが３分割の中央部となる。

【０１８７】

上部と同様に、出力結果はネイティブ層へ送られ、Ｓ１７０７ではファイルへ出力される。

【０１８８】

最後に、３分割での出力の下部の拡大について説明する。ここはループ処理の３回目（ｉ＝２）にあたる。

【０１８９】

Ｓ１５０４では、上記の中央部と同様、ｔｒａｎｓｌａｔｅ（　）メソッドを用いて元のＣａｎｖａｓの画像１４０１を上方へ平行移動する設定を行う。平行移動の距離はＨｂｐ×２である。その移動量がＨｂｐ×２である理由は、上部での出力で元画像のＨｂｐの高さ相当の画像が、中央部の出力で元画像のＨｂｐの高さ相当の画像が、それぞれ拡大されて出力済みであるためである。この平行移動の様子を仮想的に表したのが図１４の１４

１１である。
【０１９０】
続いて、Ｓ１５０５では上記の中央部と同様、ｓｃａｌｅ（　）メソッドを用いて拡大設定を行う。その拡大の様子を仮想的に表したのが図１４の１４１２である。
【０１９１】
続いて、Ｓ１５０６では上記の中央部と同様、ＣａｎｖａｓのｄｒａｗＩｍａｇｅ（）メソッドにより、出力画像の描画領域へ画像データを出力する。上述と同様に、図１４の１４１２の太線枠の点線部より上方のみが出力画像の描画領域へ出力される。出力の結果は、図１４の１４１３であり、３分割の下部となる。上部、中央部と同様にＳ１５０７でこれをファイルに出力する。
【０１９２】
ここではちょうど３分割したときの例を示したが、その他の場合にも対応可能である。例えば、分割した最下部の画像が、出力画像の描画領域の高さ（Ｈｐ）に満たない、つまりちょうど分割できず余りがでる場合がある。その際も上記と同様に平行移動と拡大、出力を行う。この時、最下部が出力された出力画像の描画領域の下の方は画像がない状態となる。この出力結果をネイティブ層２１８でファイルに保存するとき、この画像がない領域を省いて保存する。
【０１９３】
Ｓ１００７で、ＣＰＵ１００は、インジケータの停止をＯＳ層２１９へ依頼する。Ｓ１００８で、ＣＰＵ１００は、インジケータを停止して、ディスプレイ１０４の表示から消去する。
【０１９４】
＜プリントの詳細＞
図３のＳ２８ではレンダリングした後の画像データを用いてプリント処理を実行する。ここでは、図３のＳ２８のプリントの詳細について、図１１を用いて説明する。なお、図１１のＳ１１０１～Ｓ１１０３は、ＣＰＵ１００がネイティブ層２１８のプログラムを用いて実行し、Ｓ１１０４はプリンタ側で実行される処理である。
【０１９５】
Ｓ１１０１で、ＣＰＵ１００は、図１０のＳ１００６のバンドレンダリングにより分割拡大し、ファイルとして保存した各バンドのＲＧＢ画像をプリンタ１１２で利用可能な形式に変換する。プリンタが利用可能な形式は、ＲＧＢやＪＰＥＧ、ＣＭＹＫに加え、ＰＤＦ等のプリンタベンダーの独自フォーマット等もある。ここではそれらのうち、どのような形式であっても構わない。
【０１９６】
Ｓ１１０２で、ＣＰＵ１００は、前記設定情報とＳ１１０１で変換された画像データを基に、プリンタ１１２へ送信するコマンドを生成する。そのコマンドには画像フォーマット、個々の画像の画素数（図１４のＷｅ，Ｈｐ）、画像データを含む。プリンタ１１２へ送る先頭のコマンドには画像全体の画素数（図１４のＷｅ，Ｈｅ）を含む。Ｓ１１０３で、ＣＰＵ１００は、プリンタの利用できる通信プロトコルに則り、Ｓ１１０２で生成した複数のバンドのコマンドを、ユーザにより選択されたプリンタ１１２に対し順次送信する。以上の処理により、複数のバンドのコマンド（複数の画像データ）を用いた出力処理が実現される。
【０１９７】
プリンタ１１２は、コマンドとして送られる画像をプリンタ１１２内のメモリに蓄積する。プリンタ１１２は全ての画像が揃うまで待機する。全ての画像が揃うと、プリンタ１１２はそれらの画像を結合し、出力する１枚の画像にする。なお、各バンドのＲＧＢ画像のファイルを用いて１つの画像データを生成した後、その画像データをプリンタで利用可能な形式に変換しても良い。ただし、ネイティブ層でその処理を行うだけのメモリを使用可能な場合に限る。
【０１９８】

Ｓ１１０４で、プリンタ１１２は、上記結合した画像の印刷を実行する。

【０１９９】

なお、上記の例ではプリンタ１１２は全ての画像が揃うまでプリント動作を待機したが、逐次、分割された画像を受信する毎に印刷しても良い。印刷しながらコマンドとして送られる次の画像データを受信することにより、印刷終了までの時間を短縮することができる。また、上記の例では分割拡大した画像を一旦ファイルとして保存したが、保存せず外部デバイス（プリンタなど）へ逐次送信しても良い。こうすることで印刷終了までの時間短縮をしたり２次記憶装置へアクセスする事なく印刷することが可能になる。

【０２００】

従って以上説明した実施形態によれば、ハイブリッドアプリケーションにおいてスクリプト層ではメモリ不足が生じ、画像データの拡大処理が行えない環境においても、画像データの分割処理により拡大可能をとすることができる。これにより、高品位な画像をプリントすることができる。

【０２０１】

［実施形態２］

ここでは、分割拡大（バンドレンダリング）を全てネイティブ層で行う例について説明する。この実施形態は前述の実施形態１と同じく図３のステップを実現するものであり、実施形態１と処理が重複する部分が多い。そのため、説明を簡潔にするため、実施形態１との重複部分についての説明は省略し、ここでは、この実施形態に特徴的な部分のみを説明する。

【０２０２】

この実施形態の特徴的な部分は、図３におけるＳ２７のレンダリングのみである。実施形態１ではレンダリングの多くをスクリプト層２１７とスクリプト層２１７から依頼されたＯＳ層２１９において処理したが、この実施形態では、レンダリングをネイティブ層２１８が担う構成とし、高速な処理を実現している。

【０２０３】

図１６は実施形態２に従うレンダリング処理を示すフローチャートである。なお、図１６において、実施形態１において図１０で説明したのと同じステップには同じステップ参照番号を付し、その説明は省略する。図１６において、Ｓ１００１、Ｓ１００３、Ｓ１００４ＡはＣＰＵ１００がスクリプト層２１７のプログラムを実行する処理であり、Ｓ１００２とＳ１００８はＣＰＵ１００がＯＳ層２１９で実行する処理である。また、Ｓ１００５Ａ、Ｓ１００６Ａ、Ｓ１００６ＢはＣＰＵ１００がネイティブ層２１８のプログラムを実行する処理である。

【０２０４】

Ｓ１００１～Ｓ１００３の後、Ｓ１００４ではＣＰＵ１００はネイティブ層２１８に対してレンダリング依頼を行う。この際、スタンプなどベクターデータで画像を保持しているオブジェクトについてはスクリプト層２１７で拡大処理を行い、描画される。描画されたスタンプなどの画像データはＢＡＳＥ６４形式でネイティブ層２１８へ渡され、ネイティブ層２１８でこれらはバイナリデータに変換される。この際、スタンプの描画は一つずつ実行されてネイティブ層２１８へ渡され、ネイティブ層２１８でこれをバイナリデータに変換し、その後に次のスタンプの描画へ移る。このようにして、文字列データとして保持する画像を１つだけにすることができる。

【０２０５】

上述のように画像データを文字列データで保持すると、バイナリデータと比べて、多くのメモリ容量を必要とする。そのため、以上のような処理によって使用メモリ容量を削減することができる。なお、ネイティブ層２１８は受け取った画像データを一旦、２次記憶装置へファイルとして保存し、使用するときにそのファイルをオープンして使用するなどしても良い。こうすることで一時的に使用するメモリ容量を削減できる。

【０２０６】

また、ここではスクリプト層２１７からネイティブ層２１８へ各オブジェクト（写真画

像やスタンプなど）の位置情報を送る。位置情報は各オブジェクトの相対的な位置とサイズ、前面背面の順序であり、ＵＩ表示されてる配置と同様のものである。その位置情報はネイティブ層２１８で解釈され、各オブジェクトの拡大後の位置とサイズが決定される。

【０２０７】

Ｓ１００５Ａでは出力画像の描画領域の容量を算出する。これは基本的に図１０のＳ１００４と同様の処理であるが、ここでは次の制約を設ける。

【０２０８】

外部デバイスであるプリンタ１１２から受信したプリンタ情報には、実施形態１で説明したように、前述したＸＭＬの５行目に＜memory receive= 7680000 /＞という記載がある。これは、プリンタ１１２が一度に受信可能な最大データ量は７６８００００［Ｂｙｔｅ］（＝７，６８［ＭＢ］）であることを示す。従って、出力画像の描画領域の容量は、この数値以下にする必要がある。実際には、画像データ前後にヘッダやフッタを付加してプリンタ１１２へ送信するため、画像領域としてはその分少ない容量となる。なお、受信可能な最大データ量に限らず、プリンタがＲＧＢに展開して保持可能な画像のサイズをプリンタから取得し、後述の描画領域の容量を決定しても良い。

【０２０９】

Ｓ１００６Ａでは、出力画像の描画領域を確保する。その描画領域はネイティブ層２１８が管理するメモリ領域にＳ１００５Ａで算出された容量だけ確保する。Ｓ１００６Ｂではバンドレンダリングを実行する。その詳細は後述する。

【０２１０】

その後の処理は、図１０を参照して実施形態１で説明した通りである。

【０２１１】

図１７は実施形態２に従うバンドレンダリング処理の詳細を示すフローチャートである。このバンドレンダリングはネイティブ層２１８で実行され、図１７を図１５と比較すると分かるように、実施形態１ではスクリプト層２１７で実行したのと同様な処理が実施形態２ではネイティブ層２１８で実行されている。従って、ここでは、実施形態２に特徴的な部分のみ説明する。

【０２１２】

Ｓ１７０１ではＳ１５０１と同様、分割数を算出する。この分割数は出力画像の描画領域のサイズによって定められる。そのサイズは外部デバイスであるプリンタ１１２から受信したプリンタ情報に含まれる値に基づいて決定しているため、分割数も外部デバイスから受信した（又は、指定された／指示された）値に基づいて定めていると言える。

【０２１３】

Ｓ１７０２では拡大率と各画像位置を決定する。ここではファイルから開いた写真画像のサイズを外部デバイスへの入力のサイズへ変換する率を算出する。その算出には上述した位置情報を用いる。尚、拡大率と表現しているが、倍率が１を下回り、縮小になる場合もある。位置は、分割拡大後の位置を変数ｉと上記位置情報により求める。写真画像以外のスタンプの画像データの配置も前記位置情報から算出する。スタンプ画像はスクリプト層２１７で拡大処理がなされるので、ネイティブ層２１８で拡大する必要はない。

【０２１４】

Ｓ１７０３はＳ１５０３と同様、分割数でのループ処理を開始する。

【０２１５】

Ｓ１７０４では、決定した拡大率と位置の情報に基づいて出力画像の描画領域へ画像データを拡大しながら展開配置し、前面背面の順序に従って合成する。この際、スタンプも同様に配置、合成されるが拡大の必要はない。

【０２１６】

Ｓ１７０５では、出力画像の描画領域へ出力された分割拡大画像を随時、外部デバイス（プリンタ１１２）へ送信する。この際、送信データのヘッダには画像データのサイズを付加する。その送信は外部デバイス通信部２２１を介し、外部デバイスが対応しているプロトコルに従う。プリンタ１１２はその分割画像を受け取ると、随時印刷を行う。ネイテ

ィブ層２１８では次の分割画像の送信をする際、プリンタ１１２へ次の分割画像の受信が可能かどうかの問い合わせを行う。プリンタ１１２はその問い合わせに応答して次の分割画像を受信可能か否かをネイティブ層に通知する。以上のようにして、随時分割画像の送信、受信、印刷が実行される。

【０２１７】

Ｓ１７０６では分割数分のバンドレンダリングが全て終了したかをどうかを調べる。ここで、未終了であれば、処理はＳ１７０４へ戻り、全て終了であればバンドレンダリング処理を終了する。

【０２１８】

従って以上説明した実施形態に従えば、外部デバイスが受信可能なデータ量を受信し、その量も考慮して、出力画像の描画領域サイズを算出してこれをメモリに確保するので、メモリ不足を発生させないようしつつ画像データの拡大処理を実行することができる。また、外部デバイスでのメモリオーバフローも防止することが可能となる。

【０２１９】

また、この実施形態に従えば、ネイティブ層で拡大処理を実行するので、スクリプト層で写真画像を拡大するよりも高速に画像の拡大処理を実行でき、処理時間を短縮することができる。

【０２２０】

なお、本願ではネイティブ層にてプリンタ用のコマンドを送信しているが、ＯＳ層で送信しても良い。

【０２２１】

［さらに別の実施形態］

また、図１で示した情報処理装置の例は、携帯可能なモバイルコンピュータ（スマートフォンやタブレット端末）を想定したハードウェアとなっているが、本発明はそれに限定されない。例えば、据え置き型のパーソナルコンピュータ、サーバ、ゲーム機、デジタルカメラ等のハードウェア上においても、同様の構成を実現することは可能である。

【０２２２】

加えて、上記の実施形態では外部デバイスとしてプリンタの例を説明したが、本発明はこれに制限されるものではない。例えば、外部デバイスとして、他のスマートフォンやタブレット端末、ＰＣやサーバ、ゲーム機なども対象となる。

【０２２３】

さらに加えて、上記実施形態では、コンテンツ（写真画像やスタンプ画像）の描画として、ＪａｖａＳｃｒｉｐｔのＣａｎｖａｓ機能で例に挙げて説明しているが、コンテンツの描画は、これに限定されるものではない。例えば、ＳＶＧ（Ｓｃａｌａｂｌｅ Ｖｅｃｔｏｒ Ｇｒａｐｈｉｃｓ）を利用して、コンテンツを描画することも可能である。

【０２２４】

また、以上説明した実施形態では、デバイス内の画像フォルダから画像を１枚選択した例について説明したが、本発明はこれに限定されるものではない。例えば、データの絶対パスを指定する、画像の入っているフォルダごと指定する、デバイスのカメラ機能を利用しその場で画像を撮影するなどにより画像が選択されても良い。また、データの取得先についても、インターネット上の画像を選択する、脱着可能な記憶媒体内の画像を選択する、外部のデバイスと通信で画像を取得する、などが挙げられる。

【０２２５】

加えて、上記実施形態のプリンタは、インクジェットプリンタ、レーザープリンタ、昇華型プリンタ、ドットインパクトプリンタ等を利用することができる。これらには、プリンタ機能、スキャナ機能等を有する、単一機能ではない、いわゆる、複合機（マルチファンクションプリンタ）の場合もある。

【０２２６】

尚、以上の実施形態の機能は以下の構成によっても実現することができる。つまり、本実施形態の処理を行うためのプログラムコードをシステムあるいは装置に供給し、そのシ

ステムあるいは装置のコンピュータ（またはＣＰＵやＭＰＵ）がプログラムコードを実行することによっても達成される。この場合、記憶媒体から読み出されたプログラムコード自体が上述した実施形態の機能を実現することとなり、またそのプログラムコードを記憶した記憶媒体も本実施形態の機能を実現することになる。

【０２２７】

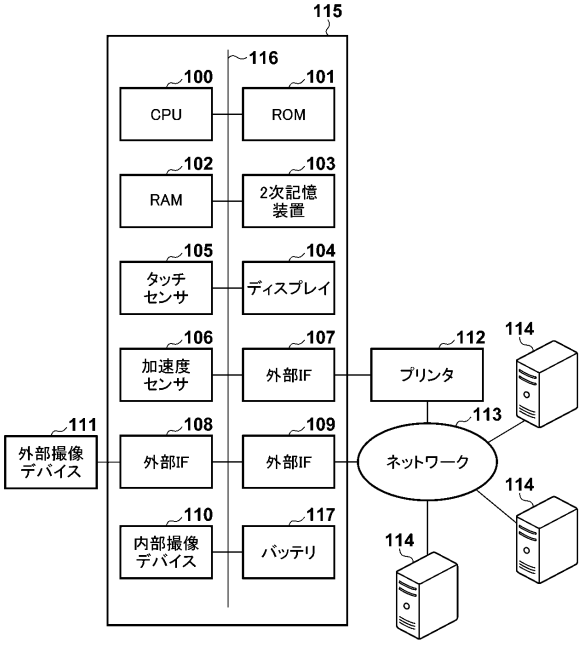
また、本実施形態の機能を実現するためのプログラムコードを、１つのコンピュータ（ＣＰＵ、ＭＰＵ）で実行する場合であってもよいし、複数のコンピュータが協働することによって実行する場合であってもよい。さらに、プログラムコードをコンピュータが実行する場合であってもよいし、プログラムコードの機能を実現するための回路等のハードウェアを設けてもよい。またはプログラムコードの一部をハードウェアで実現し、残りの部分をコンピュータが実行する場合であってもよい。

【符号の説明】

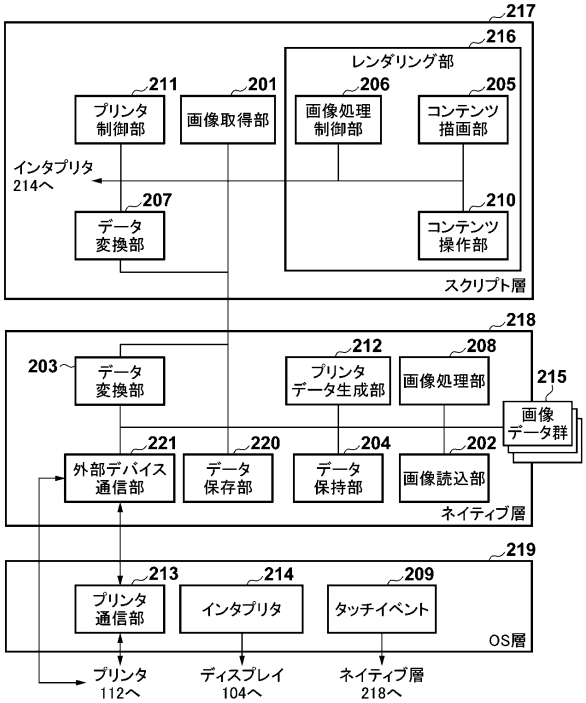
【０２２８】

２０１：画像取得部、２０２：画像読込部、２０３：データ変換部

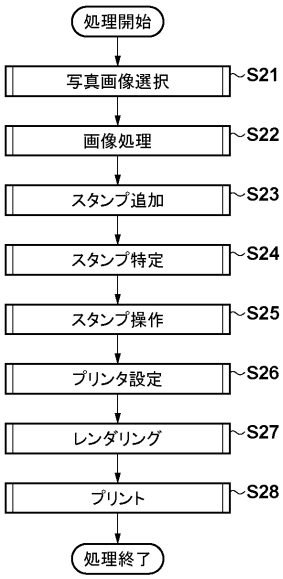
【図１】



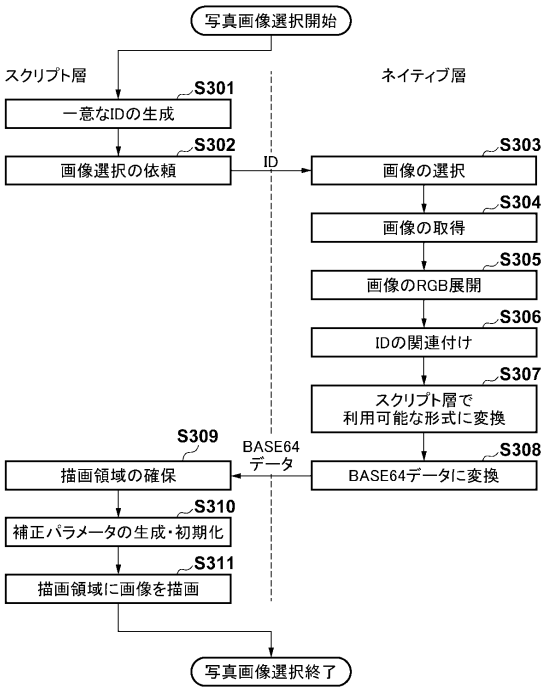
【図２】



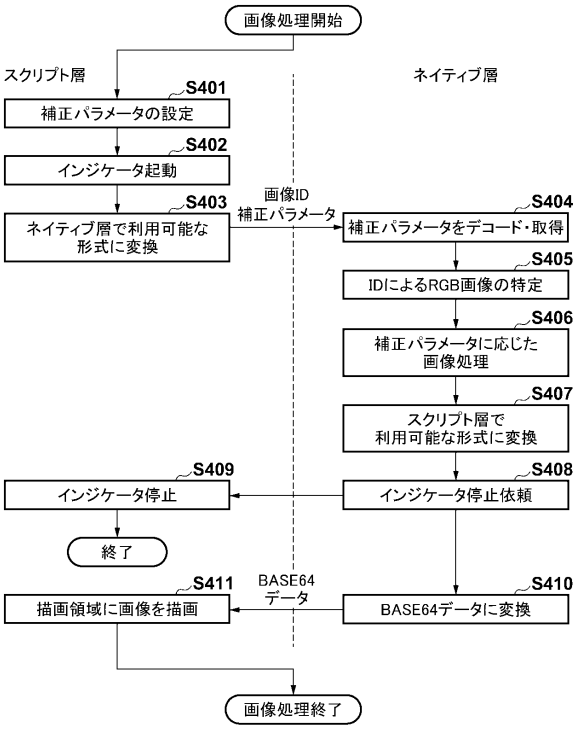
【図３】


処理開始
写真画像選択
S21
画像処理
S22
スタンプ追加
S23
スタンプ特定
S24
スタンプ操作
S25
プリンタ設定
S26
レンダリング
S27
プリント
S28
処理終了

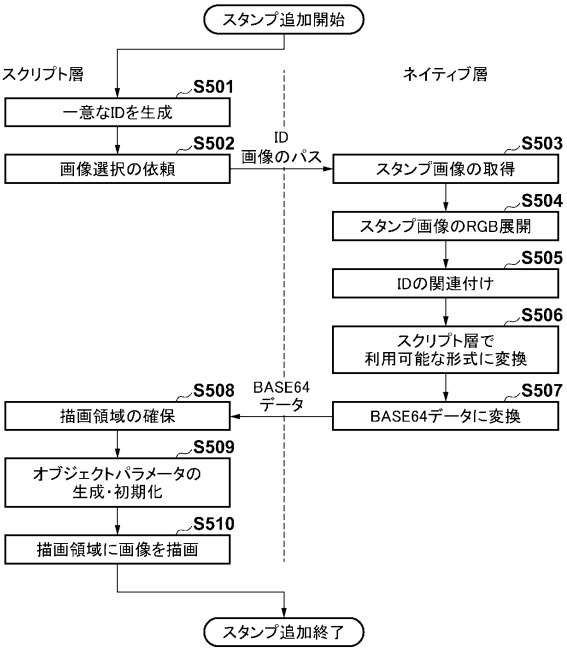
【図４】


写真画像選択開始
スクリプト層
ネイティブ層
S301
一意なIDの生成
S302
画像選択の依頼
ID
S303
画像の選択
S304
画像の取得
S305
画像のRGB展開
S306
IDの関連付け
S307
スクリプト層で
利用可能な形式に変換
S308
BASE64データに変換
BASE64
データ
S309
描画領域の確保
S310
補正パラメータの生成・初期化
S311
描画領域に画像を描画
写真画像選択終了

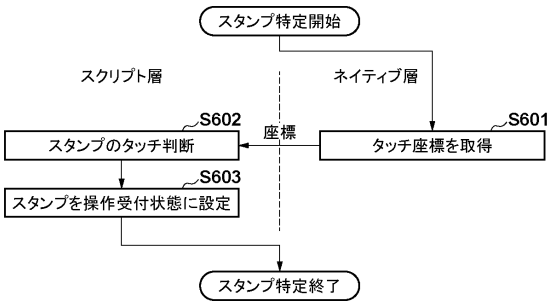
【図５】


画像処理開始
スクリプト層
ネイティブ層
S401
補正パラメータの設定
S402
インジケータ起動
S403
ネイティブ層で利用可能な
形式に変換
画像ID
補正パラメータ
S404
補正パラメータをデコード・取得
S405
IDによるRGB画像の特定
S406
補正パラメータに応じた
画像処理
S407
スクリプト層で
利用可能な形式に変換
S408
インジケータ停止依頼
S409
インジケータ停止
終了
S411
描画領域に画像を描画
BASE64
データ
S410
BASE64データに変換
画像処理終了

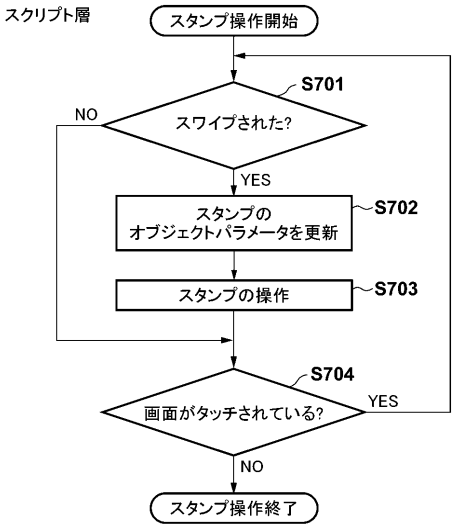
【図６】


スタンプ追加開始
スクリプト層
ネイティブ層
S501
一意なIDを生成
S502
画像選択の依頼
ID
画像のパス
S503
スタンプ画像の取得
S504
スタンプ画像のRGB展開
S505
IDの関連付け
S506
スクリプト層で
利用可能な形式に変換
S507
BASE64データに変換
BASE64
データ
S508
描画領域の確保
S509
オブジェクトパラメータの
生成・初期化
S510
描画領域に画像を描画
スタンプ追加終了

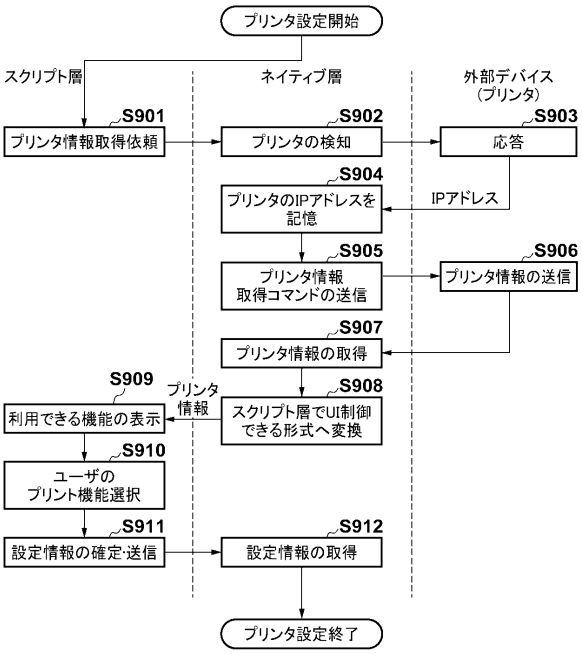
【図７】


スタンプ特定開始
スクリプト層
ネイティブ層
S602
スタンプのタッチ判断
座標
S601
タッチ座標を取得
S603
スタンプを操作受付状態に設定
スタンプ特定終了

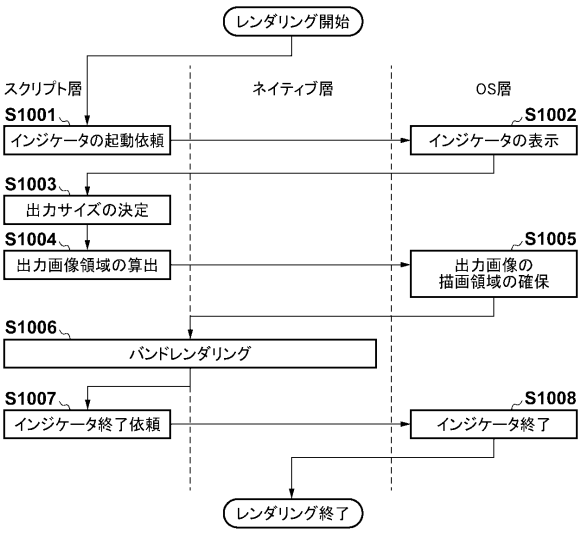
【図８】


スクリプト層
スタンプ操作開始
S701
スワイプされた?
NO
YES
スタンプの
オブジェクトパラメータを更新
S702
スタンプの操作
S703
S704
画面がタッチされている?
YES
NO
スタンプ操作終了

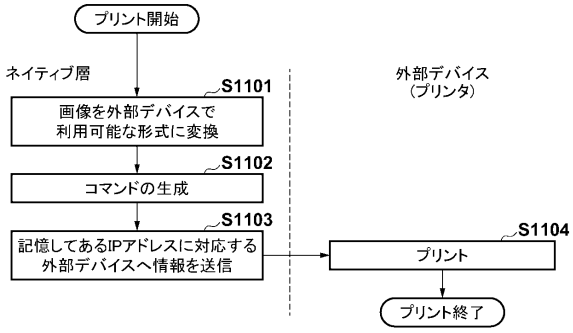
【図９】


プリンタ設定開始
スクリプト層
ネイティブ層
外部デバイス
(プリンタ)
S901
プリンタ情報取得依頼
S902
プリンタの検知
S903
応答
S904
プリンタのIPアドレスを
記憶
IPアドレス
S905
プリンタ情報
取得コマンドの送信
S906
プリンタ情報の送信
S907
プリンタ情報の取得
S909
利用できる機能の表示
プリンタ
情報
S908
スクリプト層でUI制御
できる形式へ変換
S910
ユーザの
プリント機能選択
S911
設定情報の確定･送信
S912
設定情報の取得
プリンタ設定終了

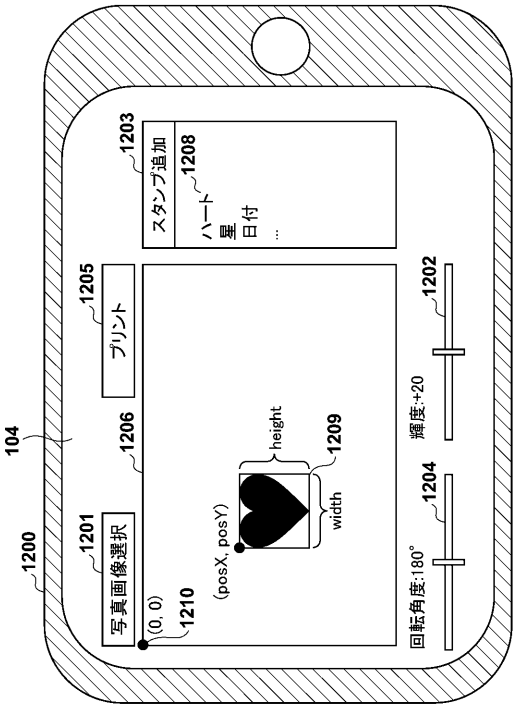
【図１０】


レンダリング開始
スクリプト層
ネイティブ層
OS層
S1001
インジケータの起動依頼
S1002
インジケータの表示
S1003
出力サイズの決定
S1004
出力画像領域の算出
S1005
出力画像の
描画領域の確保
S1006
バンドレンダリング
S1007
インジケータ終了依頼
S1008
インジケータ終了
レンダリング終了

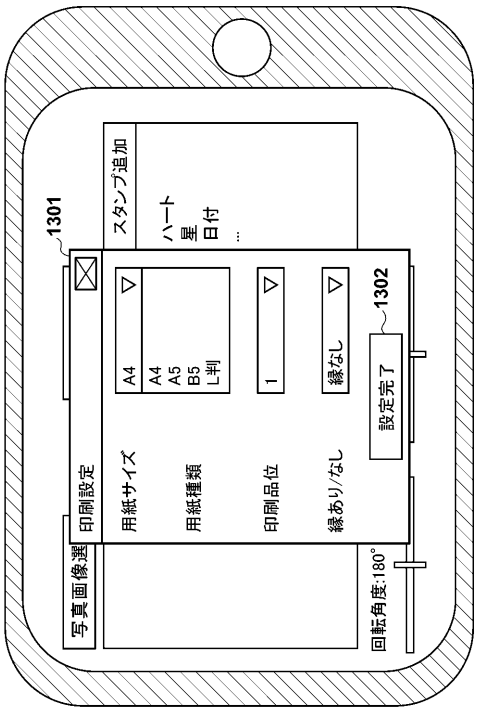
【図１１】



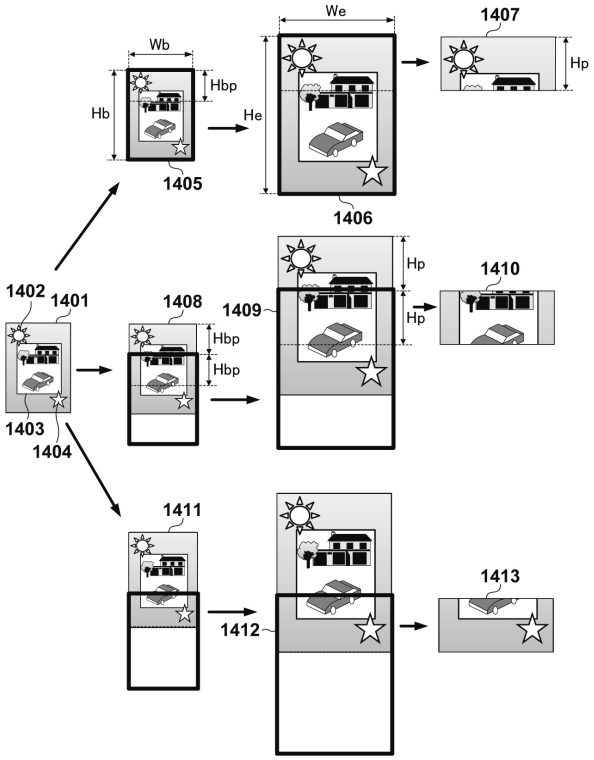
【図１２】



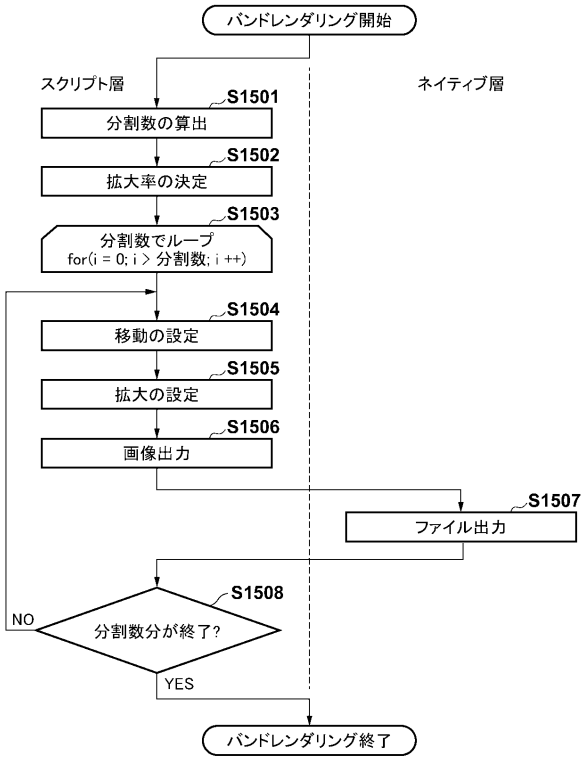
【図１３】



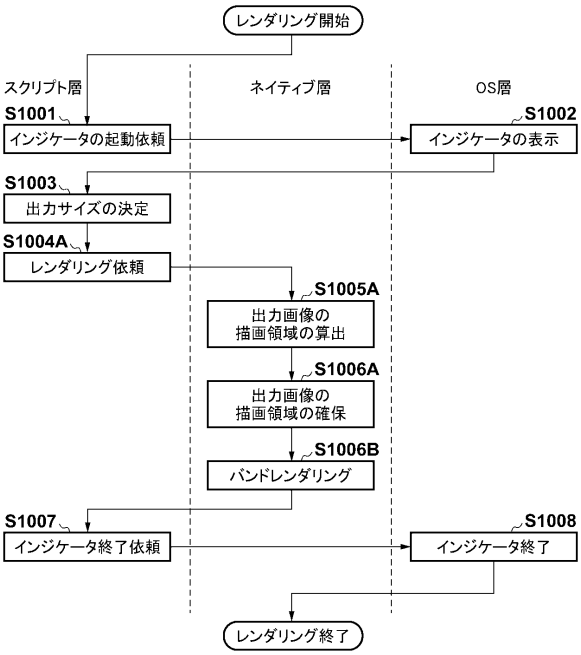
【図１４】



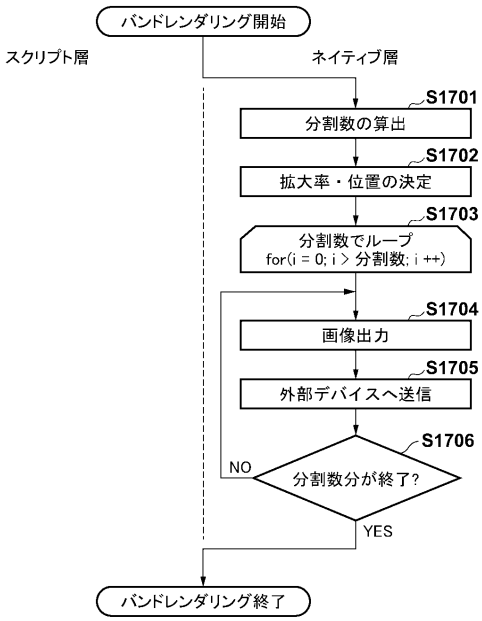
【図１５】


バンドレンダリング開始
スクリプト層
ネイティブ層
S1501
分割数の算出
S1502
拡大率の決定
S1503
分割数でループ
for(i = 0; i > 分割数; i ++)
S1504
移動の設定
S1505
拡大の設定
S1506
画像出力
S1507
ファイル出力
S1508
分割数分が終了?
NO
YES
バンドレンダリング終了

【図１６】


レンダリング開始
スクリプト層
ネイティブ層
OS層
S1001
インジケータの起動依頼
S1002
インジケータの表示
S1003
出力サイズの決定
S1004A
レンダリング依頼
S1005A
出力画像の
描画領域の算出
S1006A
出力画像の
描画領域の確保
S1006B
バンドレンダリング
S1007
インジケータ終了依頼
S1008
インジケータ終了
レンダリング終了

【図１７】


バンドレンダリング開始
スクリプト層
ネイティブ層
S1701
分割数の算出
S1702
拡大率・位置の決定
S1703
分割数でループ
for(i = 0; i > 分割数; i ++)
S1704
画像出力
S1705
外部デバイスへ送信
S1706
分割数分が終了?
NO
YES
バンドレンダリング終了

フロントページの続き

(72)発明者　鷲見　尚紀
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内
(72)発明者　鈴木　智博
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内
(72)発明者　梅田　清
東京都大田区下丸子３丁目３０番２号　キヤノン株式会社内

審査官　橘　高志

(56)参考文献　特開２０１２－１６８６３７（ＪＰ，Ａ）
特開２００８－０９０３７７（ＪＰ，Ａ）
特開２０１２－２３８２２０（ＪＰ，Ａ）
特開２０１２－１０８８２５（ＪＰ，Ａ）
特開２０１１－１０７３７８（ＪＰ，Ａ）
特開２００７－０５８７８３（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｈ０４Ｎ　　１／３９３
Ｂ４１Ｊ　　５／３０
Ｂ４１Ｊ　２１／００
Ｂ４１Ｊ　２９／３８
Ｇ０６Ｆ　　３／１２
Ｇ０６Ｔ　　３／４０