(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년09월05일
(11) 등록번호 10-1651309
(24) 등록일자 2016년08월19일

(51) 국제특허분류(Int. Cl.)
G06F 9/44 (2006.01) *G06F 7/22* (2006.01)
G06F 9/345 (2006.01) *G06F 9/38* (2006.01)
(21) 출원번호 10-2011-7019746
(22) 출원일자(국제) 2010년02월05일
심사청구일자 2015년01월08일
(85) 번역문제출일자 2011년08월25일
(65) 공개번호 10-2011-0129382
(43) 공개일자 2011년12월01일
(86) 국제출원번호 PCT/US2010/023256
(87) 국제공개번호 WO 2010/098959
국제공개일자 2010년09월02일
(30) 우선권주장
12/393,950 2009년02월26일 미국(US)
(56) 선행기술조사문헌
US6377688 B1
US6437803 B1
US20030188195 A1
US20040030894 A1

(73) 특허권자
마이크로소프트 테크놀로지 라이센싱, 엘엘씨
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
아브도 나딤 와이
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 인터내셔널 페이턴츠 마이크로소프트 코포레이션
알부 보이쿠 안톤
미국 워싱턴주 98052-6399 레드몬드 원 마이크로소프트 웨이 인터내셔널 페이턴츠 마이크로소프트 코포레이션
(74) 대리인
제일특허법인

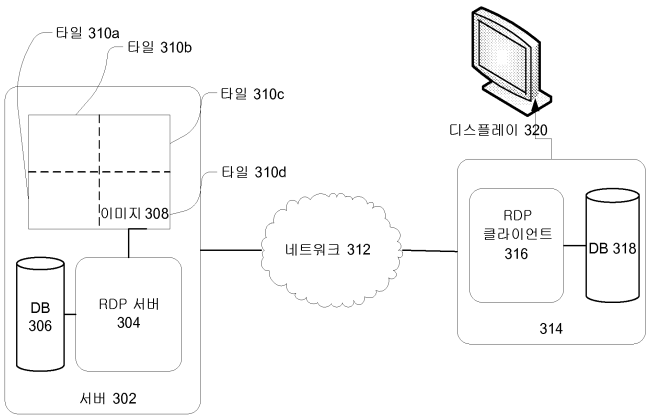
전체 청구항 수 : 총 20 항 심사관 : 권현수

(54) 발명의 명칭 **이미지 분류 방법 및 시스템**

(57) 요 약

RDP(remote desktop protocol)에 대한 벡터화 타일 차별화 알고리즘을 위한 시스템, 방법 및 컴퓨터 판독 가능 매체가 개시되어 있다. 서버는 타일을 식별하는 빅 키(big key)를 생성하는 데 이용되고 이 빅 키에 대한 추적을 유지하는 CBC-변수 벡터화 해시 알고리즘을 실행한다. 알고리즘의 직렬 버전이 한 번에 이미지의 단일 부분-32비트 등-에 대해 작용하는 경우, 벡터화 알고리즘은 동시에 복수의 이러한 부분에 작용한다. 빅 키와 부합하는 제 2 빅 키를 갖기 때문에 타일이 RDP를 통해 클라이언트에게 이미 전송된 것을 서버가 식별하는 경우에, 서버는 클라이언트에게 빅 키를 전송-수신된 타일을 캐시화함-하고, 클라이언트는 디스플레이될 적절한 타일을 액세스하기 위해 그것을 이용한다. 서버가 클라이언트에게 아직 전송되지 않은 타일을 식별하는 경우, 서버는 클라이언트에게 타일을 전송한다.

대 표 도



명 세 서

청구범위

청구항 1

병렬 처리 시스템의 프로세서가 한 번에 처리할 수 있는 비트 수에 대응하는 비트 폭을 갖는 상기 병렬 처리 시스템 상에서 이미지를 분류하는 컴퓨터 구현 방법으로서,

상기 방법은 제 1 임의 숫자(random number), 상기 제 1 임의 숫자와 상이한 제 2 임의 숫자, 데이터(datum), 제 1 키 및 제 2 키와 관련되고,

상기 방법은,

상기 병렬 처리 시스템에 의해, 상기 이미지를 두 개의 그룹으로 분할하는 단계- 각각의 그룹은 상기 이미지 중 상기 비트 폭과 동일한 비트 수를 포함함 -와,

상기 병렬 처리 시스템에 의해, 각각의 그룹에 대해 해시(hash)를 실행하는 단계와,

상기 해시를 실행하는 단계 이후에, 상기 병렬 처리 시스템에 의해, 컴퓨터 메모리 내에 상기 제 1 키 및 상기 제 2 키를 포함하는 빅 키(big key)를 저장하는 단계

를 포함하되,

상기 해시를 실행하는 단계는,

상기 데이터의 새로운 값을, 상기 그룹의 바이너리에 표현된 값과 상기 제 1 임의 숫자를 곱하고 상기 제 2 임의 숫자를 합산한 상기 데이터의 현재 값의 합계와 같게 되도록 설정하는 단계와,

상기 제 1 키의 새로운 값을 상기 데이터에 의해 수정된 상기 제 1 키의 현재 값과 같게 되도록 설정하는 단계와,

상기 제 2 키의 새로운 값을 상기 데이터에 의해 수정된 상기 제 2 키의 현재 값과 같게 되도록 설정하는 단계를 포함하는

컴퓨터 구현 방법.

청구항 2

제 1 항에 있어서,

통신 네트워크를 통해 클라이언트 컴퓨터에 상기 이미지를 전송하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 3

제 2 항에 있어서,

통신 네트워크를 통해 클라이언트 컴퓨터에 상기 이미지를 전송하는 단계는, 상기 빅 키가 상기 컴퓨터 메모리 내의 임의의 다른 빅 키와 부합하지 않는 것으로 판정하는 단계에 응답하여 상기 이미지를 상기 클라이언트 컴퓨터에 전송하는 단계를 포함하는

컴퓨터 구현 방법.

청구항 4

제 3 항에 있어서,

상기 클라이언트 컴퓨터에게 상기 빅 키를 전송하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 5

제 4 항에 있어서,

상기 클라이언트 컴퓨터에게 상기 빅 키를 사용하라는 표시를 전송함으로써, 클라이언트 해시로부터 제 2 이미지를 검색하도록 하고, 상기 클라이언트 컴퓨터가 상기 표시 대신에 상기 제 2 이미지를 수신한 것처럼 상기 제 2 이미지를 처리하도록 하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 6

제 2 항에 있어서,

상기 이미지를 상기 클라이언트 컴퓨터에게 전송하기 전에 상기 이미지를 인코딩하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 7

제 1 항에 있어서,

각각의 그룹에 대한 동작을 실행하기 전에 상기 데이터, 상기 제 1 키 및 상기 제 2 키를 각각 0으로 설정하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 8

제 1 항에 있어서,

상기 제 1 키는 상기 제 1 키와 상기 데이터에 대해 배타적 OR(exclusive-or) 연산을 실행함으로써 상기 데이터에 의해 수정되고,

상기 제 2 키는 상기 제 2 키와 상기 데이터에 대해 배타적 OR 연산을 실행함으로써 상기 데이터에 의해 수정되는

컴퓨터 구현 방법.

청구항 9

제 1 항에 있어서,

상기 이미지는 시작점과 종료점을 갖고,

각각의 그룹에 대해 실행되는 동작들은 상기 동작들이 아직 실행되지 않은 그룹 중 상기 시작점에 최근접한 그룹에 대해 먼저 실행되는

컴퓨터 구현 방법.

청구항 10

제 1 항에 있어서,

상기 비트 폭에 의해 분할된 이미지 크기가 0이 아닌 나머지 비트 수(remainder number of bits)를 갖는 것으로 판정된 것에 응답하여, 직렬 CBC(cipher block chaining) 알고리즘을 이용하여 상기 나머지 비트 수를 해싱하여 후속-제 1 키(post-first key) 및 후속-제 2 키를 생성하는 단계와, 상기 후속-제 1 키에 기초하여 상기 제 1 키를 결정하는 단계와, 상기 후속-제 2 키에 기초하여 상기 제 2 키를 결정하는 단계를 더 포함하는

컴퓨터 구현 방법.

청구항 11

제 10 항에 있어서,

상기 이미지는 종료점과 나머지 비트 수를 갖고,

상기 나머지 비트 수는 상기 두 개의 그룹 중 한 그룹의 일부가 아니고, 상기 나머지 비트 수는 상기 이미지 내에서 상기 이미지의 종료점에 위치하는

컴퓨터 구현 방법.

청구항 12

제 1 항에 있어서,

상기 해시를 실행하는 단계는 CBC(cipher block chaining) 변형을 수행하는 단계를 포함하는

컴퓨터 구현 방법.

청구항 13

제 1 항에 있어서,

상기 이미지는 스크린 스크레이프(screen scrape)를 포함하는

컴퓨터 구현 방법.

청구항 14

제 1 항에 있어서,

상기 이미지는 RDP(Remote Desktop Protocol) 내의 타일(tile)을 포함하는

컴퓨터 구현 방법.

청구항 15

비트 폭을 갖는 병렬 처리 시스템에서 이미지를 분류하는 시스템으로서,

상기 시스템은 제 1 임의 숫자, 상기 제 1 임의 숫자와 상이한 제 2 임의 숫자, 데이터, 제 1 키 및 제 2 키와 관련되고,

상기 시스템은,

프로세서와,

프로세서 실행가능 명령어를 포함하는 메모리

를 포함하고,

상기 프로세서 실행가능 명령어는 상기 프로세서에 의해 실행되는 경우, 상기 시스템으로 하여금 적어도,

상기 이미지를 두 개의 그룹으로 분할하게 하고- 각각의 그룹은 상기 이미지 중 상기 비트 폭과 동일한 비트 수를 포함함 -,

각 그룹에 대해, 해시를 실행하게 하고,

빅 키 저장부 내에 상기 제 1 키 및 상기 제 2 키를 포함하는 빅 키를 저장하게 하되,

상기 해시를 실행하는 것은 상기 이미지에 대하여 상기 제 1 키, 상기 제 2 키 및 상기 데이터를 포함하는 진행 참조(running reference)를 계산하는 것을 포함하고, 상기 계산은 상기 진행 참조, 상기 그룹, 상기 데이터, 상기 제 1 임의 숫자 및 상기 제 2 임의 숫자에 기초하여 이루어지는

이미지 분류 시스템.

청구항 16

제 15 항에 있어서,

상기 메모리는 상기 프로세서 상에서 실행되는 경우 상기 시스템으로 하여금 적어도,

상기 이미지를 인코딩하게 하고,

상기 인코딩된 이미지를 원격 데스크탑 프로토콜(RDP)에 따라 통신 네트워크를 거쳐 클라이언트로 전송하게 하는

프로세서 실행가능 명령어를 더 포함하는

이미지 분류 시스템.

청구항 17

제 15 항에 있어서,

상기 이미지의 하나의 픽셀은 32 비트를 포함하고, 상기 비트 폭은 128 비트를 포함하는

이미지 분류 시스템.

청구항 18

제 17 항에 있어서,

상기 메모리는 상기 프로세서 상에서 실행되는 경우, 상기 시스템으로 하여금 적어도,

상기 이미지를 분할하기 전에 상기 이미지는 정렬되어 있지 않은 것으로 판정하고, 직렬 알고리즘을 이용하여 제 1 정렬 경계 이전에 등장하는 상기 이미지의 일부분을 해싱하여 사전-제 1 키(pre-first key) 및 사전-제 2 키를 생성하게 하고,

상기 사전-제 1 키를 상기 제 1 키와 결합하고 상기 사전-제 2 키를 상기 제 2 키와 결합하게 하는

프로세서 실행가능 명령어를 더 포함하는

이미지 분류 시스템.

청구항 19

제 15 항에 있어서,

상기 빅 키가 상기 빅 키 저장부 내의 임의의 다른 빅 키에 부합하지 않는 경우 상기 빅 키는 상기 빅 키 저장부에 저장되는

이미지 분류 시스템.

청구항 20

컴퓨터 실행가능 명령어를 포함하는 컴퓨터 판독가능 저장 장치로서,

상기 컴퓨터 실행가능 명령어는, 병렬 처리 시스템의 프로세서가 한 번에 처리할 수 있는 비트 수에 대응하는 비트 폭, 제 1 임의 숫자, 상기 제 1 임의 숫자와 상이한 제 2 임의 숫자, 데이터, 제 1 키 및 제 2 키를 갖는 상기 병렬 처리 시스템에 의해 실행되는 경우, 상기 병렬 처리 시스템으로 하여금,

이미지를 두 개의 그룹으로 분할하는 동작- 각각의 그룹은 상기 이미지 중 상기 비트 폭과 동일한 비트 수를 포함함 -과,

각각의 그룹에 대해 해시를 실행하는 동작과,

상기 해시를 실행하는 동작 이후에, 컴퓨터 메모리 내에 상기 제 1 키 및 상기 제 2 키를 포함하는 빅 키를 저장하는 동작

을 수행하게 하되,

상기 해시를 실행하는 동작은,

상기 데이터의 새로운 값을, 상기 그룹의 바이너리에 표현된 값과 상기 제 1 임의 숫자를 곱하고 상기 제 2 임의 숫자를 합산한 상기 데이터의 현재 값의 합계와 같게 되도록 설정하는 동작과,

상기 제 1 키의 새로운 값을 상기 데이터에 의해 수정된 상기 제 1 키의 현재 값과 같게 되도록 설정하는 동작과,

상기 제 2 키의 새로운 값을 상기 데이터에 의해 수정된 상기 제 2 키의 현재 값과 같게 되도록 설정하는 동작을 포함하는

컴퓨터 판독가능 저장 장치.

발명의 설명

배 경 기 술

[0001] 일단 컴퓨터가 독립되어 있고 다른 컴퓨터와 최소 또는 적은 대화만을 한다고 해도, 오늘날의 컴퓨터는 LAN(Local Area Networks) 및 WAN(Wide Area Networks) 등과 같은 통신 네트워크를 통해 매우 다양한 다른 컴퓨터와 대화하게 된다. 인터넷™의 광범위한 성장으로, 컴퓨터들 사이의 접속성은 더 중요해지고, 여러 새로운 애플리케이션 및 기술의 가능성을 제공하였다. 대규모 네트워크의 성장 및 저렴한 퍼스널 컴퓨터의 광범위한 이용 가능성은 많은 사람들이 일하고, 대화하고, 통신하고, 오락하는 방식을 근본적으로 변화시켰다.

[0002] 그 인기가 증가하고 있는 한 형태의 네트워크는 일반적으로 RDP(Remote Desktop Protocol), ICA(Independent Computing Architecture) 등과 같은 프로토콜을 사용하여 원거리 클라이언트와 데스크탑 및 다른 애플리케이션을 공유할 수 있는 가상 컴퓨팅 시스템으로 지칭될 수 있다. 이러한 컴퓨팅 시스템은 전형적으로 키보드 입력 및 마우스 클릭 또는 선택을 클라이언트로부터 서버로 전송하여, 네트워크 접속(예를 들면, 인터넷)을 통해 다른 방향으로 다시 스크린 업데이트를 중계한다. 이와 같이 사실 상 클라이언트 장치가 단지 서버 측에 표시된 애플리케이션의 스크린샷을 전송할 때에도 사용자는 자신의 장치가 LAN의 일부분으로서 작동하는 듯한 경험을 한다.

발명의 내용

해결하려는 과제

[0003] 클라이언트에게 그래픽 데이터를 전송하는 2개의 통상적인 기술은 무엇을 어떻게 하여 대상을 그리는지에 대해 클라이언트측에서의 서브-루틴을 알려주는 그래픽 프리미티브(graphic primitive) 및 다른 연산을 전달하고, 클라이언트에게 디스플레이될 비트맵 이미지를 전달하는 것이다. 프리미티브의 시퀀스가 너무 복잡할 때, 때때로 다른 더 복잡한 프리미티브 연산의 잠재적으로 긴 시퀀스를 전송하는 것 보다는 보다 간단하게 디스플레이될 수 있는 비트맵 표현을 전달하는 것이 더 이치에 맞을 것이다. 그러나 한정된 네트워크 대역폭뿐만 아니라 최대 비트 스트림 압축기(bit stream compressors)의 한계에 기인하여 스크린의 전체 비트맵 표현을 계속적으로 전송하는 것은 너무 많은 비용을 소모할 것이다.

[0004] 이러한 문제를 완화하기 위해서, 클라이언트(애플리케이션 윈도우 등)로 전송되는 프레임은 타일(tiles)로 세분될 것이다. 다음에 이러한 타일은 클라이언트측에서 캐시화되고, 타일이 2개의 비트맵 사이에 반복될 때 클라이언트에게 타일을 재전송하는 대신에, 서버는 캐시화된 타일을 클라이언트에게 디스플레이하게 하는 인스트럭션을 전송한다. 이는 특히 타일이 자주 반복되는 경우에 RDP 세션의 대역폭 비용을 크게 감소시킬 수 있다. 그러나 이와 같이 할 때 처리 자원은 오로지 타일을 캐시화하는데 사용되어야 한다.

[0005] 또한 타일화(tiling) 알고리즘은 때때로 캐시 히트(cache hit)의 기회를 최대화하는 방식으로 구현된다. 타일이 더 작은 경우에 (해당 프레임 또는 나중의 프레임 내에서) 2회 사용될 더 많은 가능성을 갖고 있다. 타일이 너무 작은 경우에 2개의 타일 사이에서 캐시 해트로부터 매우 작은 이득만을 얻게 되기 때문에 때때로 최소의 유용한 타일 크기가 존재한다.

[0006] 이러한 RDP 비트맵 캐시화 알고리즘 및 타일 사이의 차이 검출("타일 차별화(tile differencing)")은 RDP 디스플레이 스트림의 대역폭을 LAN, WAN 또는 wLAN(wireless local area network)을 통해 전송하기에 허용 가능한 레벨까지 감소시킨다는 점에서 매우 중요하다. 이러한 캐시화 알고리즘은 전형적으로 네트워크를 통해 클라이언트에게 정보를 전송하기 위해 해당 서버에게 요구되는 대역폭의 양을 감소하는 것의 대가로 서버에서 (주로 중앙 처리 장치(CPU)의) 처리 시간을 절충한다.

[0007] RDP 비트맵 캐시화의 주요 처리 비용 중 하나는 해시 알고리즘(hash algorithm)의 사용인데, 이 알고리즘은 대형 이미지 데이터를, 어레이 또는 트리(tree) 등과 같이 분류된 데이터 구조에 대한 목차로서 사용될 수 있는 더 작은 데이터로 변환하는 것이다. 몇몇 해시화 알고리즘은 CBC(cipher block chaining) 알고리즘, 또는 CBC 알고리즘의 변종을 구현한다. 그러나 임의의 다른 자원-서버의 네트워크 대역폭 등-이 소진되기 전에 모든 이용 가능한 처리 자원이 RDP 세션에 의해 이용될 수 있기 때문에 이 해시화 알고리즘에 사용되는 처리 시간은 서버의 확장성을 저해할 수 있다. 이 처리 시간은 또한 이미지 프레임을 인코딩하는데 요구되는 시간을 이 프레임이 생성되고 클라이언트로 전달되는 레이트(프레임-레이트(FPS))만큼 증가시킨다.

[0008] CBC 해시 알고리즘이 전형적으로 SIMD(single instruction, multiple data) 프로세서 등에서와 같이 병렬 처리에 잘 맞지 않는 직렬이기 때문에 현재의 병렬 프로세서를 가지고 해시화 알고리즘의 속도를 증가하는 것은 어려운 일이다.

[0009] 그 ISA(instruction set architecture) 내에 SIMD 인스트럭션을 갖는 벡터 프로세서로 알려진 등급의 프로세서가 존재한다. NEHALEM™ 프로세서 등의 몇몇 INTEL™ x86 ISA 프로세서 내의 SSE 4.2 인스트럭션 등과 같은 SSE(Streaming SIMD extensions)는 이러한 SIMD 인스트럭션의 한 형태이다. 이러한 프로세서는 한 번에 큰 단위의 데이터에 작용할 수 있기 때문에 소정 타입의 데이터의 처리를 가속화할 수 있다. 예컨대, 이미지가 처리되는 경우에, 한 번에 단일 화소에 작용하는 것 대신에 SIMD 프로세서는 단일 인스트럭션과 병렬로 몇몇 화소에 작용할 수 있다. 이것은 인스트럭션 자체의 처리 성능을 향상시킬 뿐 아니라 메모리로부터 데이터를 인출하는데 소요되는 시간을 감소시킬 수 있다.

[0010] SIMD 인스트럭션은 압축을 위한 이미지 데이터 처리 등에서와 같이, 몇몇 유형의 프로세서의 성능을 향상시킬 가능성을 제공하는 한편, 이 프로세스를 실행하는데 필요한 알고리즘 및 기술은 비-벡터 프로세서보다 훨씬 더 어려운 것이다. 데이터 흐름 및 병렬로 작용될 수 있는 방식으로 데이터를 조직화하는 데 특별히 주의를 기울여야 한다. 그것 때문에, RDP 타일에 대한 해시화 연산의 병렬화를 증대시키는 새로운 기술의 도움을 받아야 할 것이다.

과제의 해결 수단

[0011] 본 발명은 가속화를 위해서 직렬 알고리즘이 작동하는 동안 다수의 비트에 병렬로 해시 알고리즘을 컴퓨팅한 다음, 이 개별적으로 컴퓨팅된 해시를 조합하여 이미지를 위한 해시를 결정하는 것에 의해 병렬 처리의 사용을 가능하게 한다. 이것은 충돌 히트 레이트(collision hit rate)-해시 알고리즘에 대한 2개의 서로 다른 입력이 동일한 출력을 생성할 레이트-를 아주 작은 양을 제외하고 거의 증가시키지 않으면서 처리를 위해 필요한 CPU 시간의 큰 감소를 가져온다. 많은 RDP 타일 해시화 상황에서, 이 최소의 충돌 히트 레이트 증가는 허용 가능하다.

[0012] 본 발명은 충돌 히트 레이트의 소정의 증가를 허용하면서 해시화 속도의 증가로부터 이익을 얻을 수 있는 한, RDP 타일 분류를 넘어서는 시나리오에서 유용한 것이다. 이러한 충돌 히트 레이트의 증가는 매우 작은, 대략 (100/2^192)%이다.

[0013] 본 발명의 예시적인 실시예에서, SIMD 인스트럭션을 이용한 RDP 비트맵 해시 알고리즘 가속화를 위한 방법, 시스템 및 컴퓨터 판독 가능 저장 매체가 제공된다.

[0014] 본 발명은 2개의 경우에 RDP에 대한 큰 이익을 제공한다. 첫 번째로, CBC 해시 알고리즘 또는 CBC-타입 해시 알고리즘은 비트맵 타일의 해시를 컴퓨팅하는 데 이용될 수 있다. 그 비트맵 타일이 다시 나타나면, 해시 함수는 동일한 캐시 키를 생성할 것이고 서버는 해당 비트맵을 재전송하는 것을 방지함으로써 대역폭을 절약할 수 있다.

[0015] 두 번째로, RDP가 순수한 스크린 스크랩핑(screen scraping)(다른 프로그램의 디스플레이 출력으로부터 이미지 데이터를 추출함)을 이용하여 작동하는 경우에, 이 동일한 해시 함수를 사용하여 각각의 새로운 프레임에서 이미지 타일의 해시를 컴퓨팅하여 프레임 대 프레임마다 어느 것의 타일이 변경되었는지에 대한 빠른 비교를 실행할 수 있다.

[0016] 이 모든 경우는 해시 함수가 전형적으로 RDP 스택(stack)의 최고 CPU 비용 중 하나를 차지한다는 것과, RDP가 몇몇 경우에 CPU에 한정되므로 CPU 비용의 감소는 RDP의 처리량이 증가하도록 허용할 수 있다는 것을 의미한다.

[0017] 당업자라면 본 명세서의 하나 이상의 다양한 측면은 본 명세서에 참조된 측면에 영향을 주는 회로 및/또는 프로그래밍을 포함할 수 있으나 이것으로 한정되지 않고, 회로 및/또는 프로그래밍은 사실상 시스템 설계자의 설계 선택에 기초하여 본 명세서에 참조된 측면에 영향을 주도록 구성되는 하드웨어, 소프트웨어 및/또는 펌웨어의 임의의 조합이 될 수 있다는 것을 이해할 것이다.

[0018] 상술된 내용은 요약서이고, 그에 따라 필연적으로 세부 사항의 간략화, 일반화 및 생략을 포함한다. 당업자라면 위의 요약은 오로지 예시를 위한 것이고 어떠한 방식으로 제한하도록 의도되지 않았음을 이해할 것이다.

[0019] 본 명세서의 발명에 따른 해시 가속화를 위한 시스템, 방법 및 컴퓨터 판독 가능 매체는 이하의 첨부된 도면을 참조하여 더 설명될 것이다.

도면의 간단한 설명

[0020] 도 1은 본 명세서에 설명된 해시 가속화가 구현될 수 있는 예시적인 범용 컴퓨팅 환경을 도시하는 도면.

도 2는 해시 가속화를 위한 예시적인 연산 절차를 도시하는 도면.

도 3은 상기 가속화된 타일 차별화 기법을 이용하는 RDP(remote desktop protocol)를 통해 통신하는 클라이언트 및 서버를 도시하는 도면.

발명을 실시하기 위한 구체적인 내용

[0021] 도 1은 본 명세서에 설명된 기법이 사용될 수 있는 범용 컴퓨팅 장치를 도시하는 블록도이다. 컴퓨팅 시스템 환경(120)은 단지 적당한 컴퓨팅 환경의 일례에 불과하고 본 명세서에 제시된 청구 대상의 용도 또는 기능의 범주에 대한 어떠한 제한도 제안하고자 의도된 것이 아니다. 또한 컴퓨팅 환경(120)은 예시적인 운영 환경(120)에서 도시된 구성 요소 중 어느 하나 또는 그 조합에 관련된 임의의 의존성 또는 요구 사항을 갖는 것으로도 해석되지 않아야 한다. 몇몇 실시예에서, 다양한 도시된 컴퓨팅 소자는 본 명세서의 특정 측면을 예시하도록 구성된 회로를 포함할 수 있다. 예를 들면, 본 명세서에 사용된 회로라는 용어는 펌웨어 또는 스위치에 의해 기

능을 수행하도록 구성된 특수한 하드웨어 성분을 포함할 수 있다. 다른 예시적인 실시예에서, 회로라는 용어는 기능을 수행하도록 작동 가능한 논리(logic)를 구현하는 소프트웨어 인스트럭션으로 구성되는 범용 처리 장치, 메모리 등을 포함할 수 있다. 회로가 하드웨어 및 소프트웨어의 조합을 포함하는 예시적인 실시예에서, 구현자(implementer)는 논리를 구현하는 소스 코드(source code)를 기록할 수 있고, 소스 코드는 범용 처리 장치에 의해 처리될 수 있는 머신 판독 가능 코드로 컴파일링(compiled)될 수 있다. 당업자라면 최신 기술이 하드웨어, 소프트웨어 또는 하드웨어/소프트웨어의 조합 사이의 차이가 매우 작은 정도까지 진화되었다는 사실을 인식할 것이므로, 특정 기능을 실행하는 데 있어 하드웨어 대 소프트웨어에 대한 선택은 구현자에게 남겨진 설계 선택 사항이 된다. 보다 구체적으로 당업자라면 소프트웨어 프로세스가 등가의 하드웨어 구조물로 변환될 수 있다는 것과, 하드웨어 구조물 그 자체가 등가의 소프트웨어 프로세스로 변환될 수 있다는 것을 이해할 것이다. 따라서 하드웨어 구현 대 소프트웨어 구현의 선택은 설계 선택 사항 중의 하나이고 구현자에게 선택 사항으로 남겨질 것이다.

[0022] 컴퓨터(141)는 전형적으로 다양한 컴퓨터 판독 가능 매체를 포함한다. 컴퓨터 판독 가능 매체는 컴퓨터(141)에 의해 액세스될 수 있는 임의의 이용 가능한 매체일 수 있고, 휘발성 및 비휘발성 매체, 탈착 가능 및 탈착 불가능 매체를 모두 포함한다. 시스템 메모리(122)는 ROM(read only memory)(123) 및 RAM(random access memory)(160) 등과 같이 휘발성 및/또는 비휘발성 메모리의 형태를 갖는 컴퓨터 저장 매체를 포함한다. 시동 중에서와 같이 컴퓨터(141) 내의 소자들 간에 정보를 전송하는 데 도움을 주는 기본 루틴을 포함하는 BIOS(basic input/output system)(124)는, 전형적으로 ROM(123) 내에 저장된다. RAM(160)은 전형적으로 처리 장치(159)에 즉시 액세스 가능 및/또는 처리 장치(159)에 의해 곧바로 작동되는 데이터 및/또는 프로그램 모듈을 포함한다. 제한하는 것이 아닌 예시로서, 도 1은 운영 시스템(125), 애플리케이션 프로그램(126), 다른 프로그램 모듈(127) 및 프로그램 데이터(128)를 도시한다.

[0023] 컴퓨터(141)는 또한 다른 탈착 가능/탈착 불가능, 휘발성/비휘발성 컴퓨터 저장 매체를 포함할 수 있다. 오로지 예로서, 도 1은 탈착 불가능, 비휘발성 자기 매체를 판독 또는 기록하는 하드 디스크 드라이브(138), 탈착 가능, 비휘발성 자기 디스크(154)를 판독 또는 기록하는 자기 디스크 드라이브(139) 및 CD ROM 또는 다른 광학 매체 등과 같은 탈착 가능, 비휘발성 광학 디스크(153)를 판독 또는 기록하는 광학 디스크 드라이브(140)를 도시한다. 예시적인 운영 환경에서 사용될 수 있는 다른 탈착 가능/탈착 불가능, 휘발성/비휘발성 컴퓨터 저장 매체는 자기 테이프 카세트, 플래시 메모리 카드, 디지털 다용도 디스크, 디지털 비디오 테이프, 고체 상태 RAM, 고체 상태 ROM 등을 포함하지만 이것으로 한정되지 않는다. 하드 디스크 드라이브(138)는 전형적으로 인터페이스(134) 등과 같은 탈착 불가능 메모리 인터페이스를 통해 시스템 버스(121)로 접속되고, 자기 디스크 드라이브(139) 및 광학 디스크 드라이브(140)는 전형적으로 인터페이스(135) 등과 같은 탈착 가능 메모리 인터페이스에 의해 시스템 버스(121)에 접속된다.

[0024] 상술되고 도 1에 도시된 드라이브 및 그와 연관된 컴퓨터 저장 매체는 컴퓨터(141)를 위한 컴퓨터 판독 가능 인스트럭션, 데이터 구조, 프로그램 모듈 및 다른 데이터의 저장 장치를 제공한다. 도 1에서, 예를 들면 하드 디스크 드라이브(138)는 운영 시스템(158), 애플리케이션 프로그램(157), 다른 프로그램 모듈(156) 및 프로그램 데이터(155)를 저장하는 것으로 도시되어 있다. 이러한 구성 요소는 운영 시스템(125), 애플리케이션 프로그램(126), 다른 프로그램 모듈(127) 및 프로그램 데이터(128)와 동일하거나 상이할 수 있다는 것을 유의하라. 본 명세서에서 운영 시스템(158), 애플리케이션 프로그램(157), 다른 프로그램 모듈(156) 및 프로그램 데이터(155)는 최소한도로 서로 다른 복사본을 갖는다는 것을 예시하기 위해서 서로 다른 개수로 제공되었다. 사용자는 키보드(151) 및 통상적으로 마우스, 트랙볼(trackball) 또는 터치 패드로서 지칭되는 포인팅 장치(152) 등과 같은 입력 장치를 통해 컴퓨터(141)에 커맨드 및 정보를 입력할 수 있다. 입력 장치(도시하지 않음)는 마이크로폰, 조이스틱, 게임 패드, 위성 안테나, 스캐너 등을 포함할 수 있다. 이러한 입력 장치 및 다른 입력 장치는 때때로 시스템 버스에 결합된 사용자 입력 인터페이스(136)를 통해 처리 장치(159)에 접속되지만, 병렬 포트, 게임 포트 또는 USB(universal serial bus) 등과 같은 다른 인터페이스 및 버스 구조에 의해 접속될 수도 있다. 모니터(142) 또는 다른 타입의 디스플레이 장치는 또한 비디오 인터페이스(132) 등과 같은 인터페이스를 통해 시스템 버스(121)에 접속될 수 있다. 모니터에 추가하여, 컴퓨터는 또한 출력 주변 인터페이스(133)를 통해 접속될 수 있는 스피커(144) 및 프린터(143) 등과 같은 다른 주변부 출력 장치를 포함할 수 있다.

[0025] 컴퓨터(141)는 원격 컴퓨터(146) 등과 같은 하나 이상의 원격 컴퓨터에 대한 논리 접속을 이용하여 네트워크형 환경 내에서 작동할 수 있다. 원격 컴퓨터(146)는 퍼스널 컴퓨터, 서버, 라우터(router), 네트워크 PC, 피어(peer) 장치 또는 다른 공통 네트워크 노드일 수 있고, 전형적으로 메모리 저장 장치(147)만이 도 1에 도시되어 있으나 컴퓨터(141)와 관련하여 상술된 여러 또는 모든 소자를 포함한다. 도 1에 도시된 논리 접속은

LAN(local area network : 근거리 네트워크)(145) 및 WAN(wide area network : 원거리 네트워크)(149)을 포함하지만, 다른 네트워크를 포함할 수도 있다. 이러한 네트워킹 환경은 사업체에 있어서 기업형(enterprise-wide) 컴퓨터 네트워크, 인트라넷(intranets) 및 인터넷이 보편적이다.

[0026] LAN 네트워킹 환경에서 사용될 때, 컴퓨터(141)는 네트워크 인터페이스 또는 어댑터(137)를 통해 LAN(145)에 접속된다. WAN 네트워킹 환경에서 사용될 때, 컴퓨터(141)는 전형적으로 인터넷 등과 같이 WAN(149)을 통해 통신을 형성하는 모뎀(150) 또는 다른 수단을 포함한다. 외부형 또는 내부형일 수 있는 모뎀(150)은 사용자 입력 인터페이스(136)를 통해 시스템 버스(121)에 접속되거나, 다른 적절한 메커니즘에 접속될 수 있다. 네트워크형 환경에서, 컴퓨터(141) 또는 그 일부분에 대하여 도시된 프로그램 모듈은 원격 메모리 저장 장치에 저장될 수 있다. 제한 사항이 아닌 예로서, 도 1은 메모리 장치(147)에 상주하는 원격 애플리케이션 프로그램(148)을 도시한다. 도시된 네트워크 접속은 예시적인 것이고, 컴퓨터들 사이에 통신 링크를 형성하는 다른 수단을 사용할 수 있다는 것을 이해할 것이다.

[0027] 도 2는 해시 가속화를 위한 예시적인 연산 절차를 도시한다. 직렬 해시화 프로세스는 한 번에 이미지 데이터에 대해 하나의 DWORD(double word; 여러 시스템 아키텍처에서 이것은 32비트임)를 택할 수 있는데, 여기에서 DWORD는 해당 이미지의 시작부에서 시작하여 종료부까지 적용된다. 이것은 이미지에 대한 하나의 키를 컴퓨팅할 수 있는데, 이 키는 처리되는 이미지의 각각의 새로운 DWORD에 기초하여 업데이트된다. 이와는 대조적으로, 본 발명의 예시적인 연산 절차는 한 번에 복수의 DWORDS에 작용한다. DWORD가 32비트를 포함하고, 프로세서가 동시에 128비트에 작용할 수 있는 실시예에서, 이는 데이터의 4개의 DWORDS일 수 있다. 이 절차는 이미지의 4개의 부분키를 유지-하나는 병렬로 작용하는 각각의 DWORD에 대응함-하고, 모든 이미지가 처리되어 최종 키가 생성되면 4개의 부분 키는 조합될 수 있다.

[0028] 선택 연산(202)은 제 1 임의 숫자 및 제 2 임의 숫자의 선택을 나타내고, 제 1 임의 숫자는 제 2 임의 숫자와는 상이하다.

[0029] 선택 연산(204)은 검사합(checksum), 데이터(datum), 제 1 키 및 제 2 키를 각각 0이 되도록 설정하는 것을 나타낸다. 임의의 실시예에서, 현재의 연산이 시작하기 전에 검사합, 데이터, 제 1 키 및 제 2 키는 0이므로, 그 시점에서 그들을 0으로 설정할 필요가 없다. 이 기술이 계속 합산을 실행하기 위해 4개의 숫자를 사용하는 경우에, 이 숫자를 0으로 설정함으로써 초기화될 수 있다. 검사합 및 데이터는 제 1 키 및 제 2 키를 결정하는데 사용되는 내부 변수로서 사용될 수 있다. 제 1 키 및 제 2 키는 이미지를 위한 빅 키(big key)를 포함할 수 있고, 서버가 클라이언트에게 이미지를 전송하였는지 여부를 탐색하는 것 또는 이미 수신된 이미지 대신에 빅 키를 수신하는 클라이언트가 이미지 그 자체를 탐색하는 것과 같이, 나중에 빅 키는 이미지 또는 그 참조를 탐색하는 데 이용된다. 모노리식(monolithic) 빅 키를 계산하는 것보다는 빅 키를 이러한 2개의 성분-제 1 키 및 제 2 키-으로 계산한 다음 그것을 결합하는 것이 빠르다.

[0030] 연산(206)은 이미지를 적어도 하나의 그룹으로 그룹화하는 것을 도시하고, 각각의 그룹은 비트 폭과 같은 수의 이미지의 비트 수를 포함한다. 프로세서는 한 번에 1이상의 비트의 데이터에 대해 작용할 수 있다. 예컨대, 128비트 프로세서는 한 번에 128비트의 데이터에 작용할 수 있다. 그러면 이 프로세서는 128비트의 "비트 폭"을 갖는다. 이 128비트는 128비트 수를 나타내는 것 등과 같이 동일한 데이터 구조에 속할 필요는 없다. 오히려, 이 128비트는 4개의 불연속 32비트 정수, 또는 8개의 불연속 16비트 정수 등과 같은 다수의 데이터 부분을 포함할 수 있다. 이러한 상황에서 4개의 32비트 정수의 그룹에 대해 연산이 실행될 때, 해당 연산의 결과(다수 또는 논리 시프트 등)가 128비트 내의 인접한 정수로 넘어가지 않도록 하는 인스트럭션이 존재한다.

[0031] 일 실시예에서, 이미지는 스크린 스크레이프(screen scrape)를 포함한다. 스크린 스크레이프는 다른 프로그램의 디스플레이 출력으로부터 데이터를 취득하는 것을 포함한다. 이 데이터는 출력 장치에서 디스플레이될 다른 프로그램에 의해 예정되었기 때문에, 전형적으로 그것을 해석하는데 도움이 될 문서화 또는 구조가 부족하다.

[0032] 일 실시예에서, 이미지는 RDP(Remote Desktop Protocol) 내에 타일을 포함하고, 타일은 비트맵을 포함한다. 타일은 프레임의 서브셋을 포함할 수 있고, 프레임은 전형적으로 애플리케이션 윈도우를 포함한다. 전형적인 실시예에서, 프레임은 복수의 직사각 타일로 분할되고, 오로지 변경된 타일만이 클라이언트에게 전송되어야 한다.

[0033] 이미지가 정렬 경계에서 정렬되지 않은 일 실시예에서, 제 1 비트 경계 이전의 이미지의 제 1 비트는 직렬 CBC(cipher block chaining) 알고리즘으로 해시화되어 사전-제 1 키(pre-first key) 및 사전-제 2 키(pre-

second key)를 생성한다. 예컨대, 본 연산이 128비트에 대해 병렬로 작용하고 이미지가 128비트 경계에서 정렬되지 않아서 이미지의 56비트가 이미지의 첫 번째 128비트 경계 이전에 발생하는 경우에, 위의 56비트는 직렬 CBC 알고리즘으로 해시화되고, 결과적인 사전-제 1 키 및 사전-제 2 키는 직렬 CBC 알고리즘을 통한 후속 연산의 결과적인 제 1 키 및 제 2 키와 각각 결합된다.

[0034] 연산(208)은 각각의 그룹에 있어서 해시화를 실행하는 것을 도시하는데, 이것은 검사합과 그룹을 합한 것과 같게 되도록 검사합을 설정하는 단계와, 제 1 임의 숫자에 곱해지고 제 2 임의 숫자와 합산된 데이터과 그룹을 합한 것과 같게 되도록 데이터를 설정하는 단계와, 제 1 키를 데이터에 의해 수정된 제 1 키와 같게 되도록 설정하는 단계와, 제 2 키를 데이터에 의해 수정된 제 2 키와 같게 되도록 설정하는 단계를 포함한다.

[0035] 일 실시예에서, 제 1 키는 제 1 키 및 데이터에 대해 배타적-OR(exclusive-or) 연산을 실행하는 것에 의해 데이터로 수정되고, 제 2 키는 제 2 키 및 데이터에 대해 배타적-OR 연산을 수행하는 것에 의해 데이터로 수정된다. 배타적 OR은 비트 방식 연산이고, 2개의 입력의 결과는 1이 되는데 여기에서 입력 중 하나 또는 오로지 하나의 입력 또한 1이 된다. 예컨대, "^"은 배타적 OR 연산을 배타적 OR 연산을 나타내는 것으로서, 0^0=0, 0^1=1, 1^0=1 및 1^1=0이 된다.

[0036] 이미지는 시작점과 종료점을 갖는 것으로 간주될 수 있고, 일 실시예에서 각각의 그룹에 대해 실행된 연산은 각각의 그룹에 대해 실행되는 연산은 아직 실행되지 않은 상태로 첫 번째로 시작점에 가장 가까운 그룹에 대해서부터 실행된다. 즉, 이 기법은 이미지 전체에 걸쳐 순차적으로 진행되어 한 번에 한 그룹의 비트를 취하고 다른 그룹의 비트에 대해 동일한 연산을 하기 전에 그것을 처리하는데, 이것을 이미지의 종료점에 도달할 때까지 계속한다.

[0037] 일 실시예에서 해시는 CBC(cipher block chaining) 해시를 포함한다.

[0038] 선택 연산(210)은 제 1 키의 각각의 DWORD 비트 크기에 기초하여 DWORD 제 1 키를 설정하는 것과 제 2 키의 각각의 DWORD 비트 크기에 기초하여 DWORD 제 2 키를 설정하는 것을 나타낸다.

[0039] 제 1 키 및 제 2 키가 각각 프로세서의 비트 폭보다 더 적은 개수의 비트를 포함하는 경우에, 부분 키가 계산된다. 이미지가 전부 처리되면 이러한 부분 키는 전체의 제 1 키 및 제 2 키를 결정하는 데 사용된다. 제 1 키 및 제 2 키가 각각 32비트를 포함하고 비트 폭이 128비트인 실시예를 고려한다. 이 실시예에서, 이미지의 128비트는 동시에 처리되어 4개의 32비트 부분 제 1 키 및 4개의 32비트 부분 제 2 키를 생성한다. 전체의 이미지가 처리되면, 이 4개의 부분 키를 각각 사용하여 전체의 키를 결정하거나, DWORD 제 1 키 및 DWORD 제 2 키를 결정한다.

[0040] 일 실시예에서, 4개의 부분 제 1 키를 사용하여 그들 모두에 대해 배타적 OR 연산(XOR 연산)을 실행함으로써 DWORD 제 1 키를 결정한다. 4개의 부분 제 1 키가 Key1[0], Key1[1], Key1[2] 및 Key1[3]로 표현되는 경우에, 이 연산은 논리적으로 Key1[0]^Key1[1]^Key1[2]^Key1[3]으로 표현될 수 있다.

[0041] 일 실시예에서, 4개의 부분 제 1 키는 직렬 CBC 해시 알고리즘을 이용하여 그들을 결합함으로써 DWORD 제 1 키를 결정하는 데 사용된다. 일 실시예에서, Key1[0]은 Key1[1]과 CBC 해시화되고, 그 결과는 Key1[2]와 CBC 해시화되고, 그 결과는 Key1[3]와 CBC 해시화된다.

[0042] 선택 연산(212)은 비트 폭에 의해 분할되는 이미지 크기가 0이외의 나머지 비트 수를 가질 때, 직렬 알고리즘을 가지고 나머지 비트 수를 해시화하여 후속-제 1 키(post-first key) 및 후속-제 2 키(post-second key)를 생성하는 것, 후속-제 1 키에 기초하여 제 1 키를 결정하는 것, 후속-제 2 키에 기초하여 제 2 키를 결정하는 것을 나타낸다. 이미지가 비트 폭에 의해 균일하게 분할 가능한 개수의 비트를 갖는 경우에, 상기 연산에서 설명된 바와 같은 일련의 병렬 연산은 이미지의 모든 비트마다 발생할 수 있다. 그러나 이미지가 비트 폭에 의해 고르게 분할될 수 없는 개수의 비트를 갖는 경우에, 상기 병렬 해시화와는 반대로 최종 비트의 통상적인 직렬 해시화가 발생할 수 있다. 제 1 키 및 후속-제 1 키에 대해 직렬 CBC 해시를 실행한 결과로 제 1 키를 설정하는 것 및 제 2 키 및 후속-제 2 키에 대해 직렬 CBC 해시를 실행한 결과로 제 2 키를 설정하는 것 등과 같이 결과적인 후속-제 1 키 및 후속-제 2 키는 제 1 키 및 제 2 키와 결합될 수 있다.

[0043] 일 실시예에서, 이미지는 종료점을 갖고, 이미지의 어떤 부분도 나머지 비트의 수보다 종료점에 더 가깝지 않다. 즉, 나머지 비트는 이미지의 최종 비트로서 간주될 것이고, 이들은 모든 선행하는 비트가 한 번에 비트의 비트 폭 수까지 병렬 방식으로 처리된 이후에 직렬 방식으로 처리될 것이다.

[0044] 일 실시예에서, 직렬 처리로부터 계산되는 제 1 키 및 제 2 키는, 각각의 제 2 키에 대한 직렬 CBC 해시뿐만 아

니라 각각의 제 1 키에 대한 직렬 CBC 해시, 각각의 제 2 키 및 각각의 제 1 키에 대한 XOR 연산 등을 통해서 상기 병렬 처리로부터 계산되는 제 1 키 및 제 2 키와 결합된다.

[0045] 연산(214)은 빅 키 저장부 내에 제 1 키 및 제 2 키를 포함하는 빅 키를 저장하는 것을 나타낸다. 해시화되면, 이미지는 트리 등과 같은 메모리 어드레스 및 데이터 구조에 저장된 다음, 데이터를 사용하여 메모리 어드레스의 위치에 있는 이미지를 액세스할 수 있다. 예컨대, 어레이 메모리 구조가 사용되는 경우에, 데이터는 해당 어레이의 인덱스로서 기능할 수 있고, 이미지는 어레이의 데이터-인덱스에 저장된다.

[0046] 각각의 키가 32비트를 포함하는 경우에, 빅 키는 64비트를 포함하므로 빅 키에 대한 2^64개의 가능한 값이 존재한다. 일 실시예에서, 충돌 가능성이 있기는 하지만(2개의 서로 다른 이미지 타일은 동일한 빅 키에 대응함), 그것이 발생할 여유가 너무 작아서(임의의 2개의 빅 키 사이에서 2^64 중 대략 1), 충돌이 무시되고 부정확한 이미지가 사용될 수 있음을 검출하고 정확한 타일 이미지를 식별할 이익보다 충돌을 검출하는데 요구되는 처리 리소스가 더 중요하게 된다.

[0047] 연산(216)은 이미지의 인코딩을 나타낸다. 일 실시예에서 이것은 RLE(run-length encoding)을 통해 이미지를 인코딩하는 것을 포함한다. 일 실시예에서 이것은 서로 다른 코덱(codecs)으로 이미지의 서로 다른 부분을 인코딩하는 것을 포함할 수 있다. 예컨대, 내과의사에 의해 사용되는 의학용 X-레이 등과 같은 경우 em등과 같이 이미지의 부분이 무손실 특성으로 디스플레이되어야하는 경우에, 이것은 무손실 코덱으로 인코딩될 수 있을 것이다. 그 이미지의 다른 부분은 흑색 문자이거나 백색 배경일 수 있고, RLE 알고리즘으로 그것을 인코딩하는 것은 (무손실성을 갖게될 뿐만 아니라) 그 공간을 최대로 감소할 수 있을 것이다. 이 경우에 이 2개의 코덱은 이미지의 적절한 부분에 사용될 것이다. 일 실시예에서 이미지의 색상 채널은 분리되는데, RGBA 이미지는 그 개별적인 적색(R), 녹색(G), 청색(B) 및 문자(A : alpha) 색상 채널로 분리되고, 각각의 이 채널은 별도로 인코딩된다.

[0048] 연산(218)은 통신 네트워크를 통해 클라이언트 컴퓨터로 이미지를 전송하는 것을 나타낸다. 이것은 이미지가 이미 클라이언트에게 전달되고, 클라이언트가 수신된 이미지를 캐시화(caches)한 경우에는 필수적이지 않다. 이러한 경우에 클라이언트는 빅 키를 전송할 수 있다. 클라이언트는 현재의 연산을 실행하는 장치에 의해 유지된 해시 테이블과 유사하게 수신된 이미지의 해시 테이블을 유지할 수 있다. 이와 같이 빅 키를 수신한 상황에서 클라이언트는 그 해시 구조에 대한 인덱스로서 빅 키를 사용하고, 대응하는 이미지를 수신하며, 데이터가 아닌 그 이미지를 수신한 것처럼 그 이미지를 처리할 수 있다.

[0049] 도 3은 상기 가속화된 타일 차별화 기법을 사용하는 RDP(remote desktop protocol)을 통해 통신하는 클라이언트 및 서버를 나타낸다. 서버(302)는 RDP 서버(304), 키 데이터베이스(306) 및 적어도 하나의 비트맵 이미지(308)를 포함한다. 이미지(308)는 적어도 하나의 타일(310)로 분할되고, 각각의 타일은 비트맵을 포함한다. 서버(302)는 통신 네트워크(312)를 통해 클라이언트(314)와 통신한다. 클라이언트(314)는 RDP 클라이언트(316) 및 타일 데이터베이스(318)를 포함하고, 디스플레이 장치(320)에 접속된다.

[0050] 클라이언트(314) 및 서버(302) 사이의 RDP 세션에서, 서버는 서버(302)가 실행하는 처리에 대응하는 클라이언트 이미지 정보를 네트워크(312)를 통해 전송한다. 예컨대, 클라이언트(314)는 서버에 대해 텍스트 편집기(text editor)를 실행하는 RDP 세션을 가질 수 있다. 클라이언트(314)는 현재의 편집 위치에서 텍스트 편집기에 일련의 문자를 입력하라 또는 새로운 파일을 열어라 등과 같은 명령어를 서버(302)로 전달한다. 이 명령어는 서버(302)에서 처리되고, 결과적인 디스플레이 출력은 디스플레이 장치(320)에서 디스플레이되도록 클라이언트(314)에게 되전송된다. 이러한 실시예에서, 이미지(308)는 주어진 시점에서 텍스트 편집기의 애플리케이션 윈도우를 포함할 수 있다. 사용자가 애플리케이션 윈도우의 하부 부근에 새로운 텍스트를 추가한다면, 애플리케이션 윈도우의 상부 부분은 가까운 미래에는 변경되지 않을 것이다. 그러므로 이미지(308)는 타일(310)로 분할될 수 있고, 시간에 걸쳐 복수의 이미지(308)가 클라이언트(314)에게 전송될 때, 각각의 이미지(308)는 복수의 타일(310)로서 전송되고, 오로지 모든 선행하는 타일(310)과는 상이한 타일(310)만이 전송되기만 하면 된다. 클라이언트(314)는 타일 캐시(318)에서 이전에 수신된 타일(310)을 캐시화할 수 있고, 타일(310)이 반복되는 경우에 서버(302)는 타일(310) 그 자체보다는 타일의 표시를 클라이언트(314)에게 전송할 수 있다. 이제 타일 캐시(318)에서 캐시화된 그 제 1 타일은 반복된 타일과 동일한 위치에 있지 않을 수 있다. 예컨대, 텍스트 편집 세션 내의 새로운 문서에서 대부분의 타일은 단지 백색 공간에 불과할 수 있으므로, 이 순수한 백색 타일 모두에 대해 하나의 순수한 백색 타일이 여러 번 사용될 수 있을 것이다.

[0051] 서버(302)가 이미지(308)에 대한 요청을 처음으로 수신한 때, 제 1 타일(310)을 취할 것이고, 빅 키를 결정하기 위해 그것을 해시화할 것이다. 서버(302)가 도 2에 도시된 기법에 따라서 타일(310)을 해시화하는 경우에, 키

는 제 1 키 및 제 2 키를 포함할 것이다. 다음에 RDP 서버(304)를 이용하여 이미지의 어떤 부분에 타일(310)이 속하는지를 나타내는 표시와 함께 타일을 RDP 클라이언트(316)에게 전송할 것이며, 또한 키 데이터베이스(306) 내에 해당 타일에 대한 빅 키를 저장할 것이다. 키 데이터베이스는 트리 등과 같이 정수를 저장하는 다양한 데이터 구조를 포함할 수 있다.

[0052] 각각의 연속적인 타일(310)에 있어서, 서버(302)는 그에 대한 빅 키를 결정한 다음, 키 데이터베이스(306)에 대해 비교할 것이다. 키 데이터베이스(306) 내에 부합되는 것이 없으면, 그것은 동일한 이미지를 나타내는 타일이 클라이언트(314)에게 앞서 전송된 적이 없다는 것을 의미하고, 서버(302)는 그 타일을 클라이언트(314)에게 전송하고, 이전과 같이 그 빅 키를 키 데이터베이스(306) 내에 저장할 것이다. 클라이언트는 RDP 클라이언트(316)에서 해당 타일을 수신하고, 빅 키와 함께 그것을 타일 캐시(318) 내에서 캐시화한다. 빅 키는 해시 테이블에 대한 인덱스의 역할을 하는 것 등과 같이 타일 캐시(318) 내에 이미지의 위치를 정하는 데 사용될 수 있다.

[0053] 키 데이터베이스(306) 내에 이 타일에 부합되는 것이 존재하는 경우, 이는 서버(302)가 이 타일과 동일한 이미지를 나타내는 타일을 이미 전송하였다는 것을 의미한다. 그러므로, 클라이언트(314)에게 그 타일(310)을 전송하는 것 대신에, 네트워크 리소스를 아끼기 위해 서버(302)는 최소량의 데이터를 포함하는 대응하는 빅 키를 클라이언트(314)에게 전송할 것이다. 제 1 키 및 제 2 키가 각각 32비트의 데이터를 포함하는 경우에, 빅 키는 64비트를 포함할 것이다. 클라이언트(314)는 RDP 클라이언트(316)에서 이 키 및 데이터를 수신하고, RDP 클라이언트(314)는 이 빅 키를 이용하여 타일 캐시(318) 내에 대응하는 타일의 위치를 정한다.

[0054] RDP 클라이언트(314)가 타일 그 자체를 수신하였든지, 대응하는 빅 키를 수신하였든지, 타일 캐시(318) 내에서 타일을 탐색하였든지, 이제는 타일을 갖고 있게 되었다. 타일 또는 빅 키와 함께, RDP 클라이언트(316)는 이미지(308)의 어떤 부분을 타일(310)이 포함하는지에 대한 표시를 수신할 것이다. 그러면 RDP 클라이언트(316)는 클라이언트 디스플레이 장치(320)에서 이미지의 적절한 부분 내에 타일(310)을 디스플레이할 것이다.

[0055] 견본 인스트럭션

[0056] 이하의 내용은 고급 프로그래밍 언어인 C형 구문으로 이루어진 견본으로서의 유사 코드를 포함하는데 이것은 벡터 프로세서에 실행될 때 본 발명의 실시예에 설명된 방식으로 도 2의 세부 설명으로 나타낸 것과 유사하게 이미지에 작용할 것이다.

```
__inline void __fastcall NextCBC64_SSE(
        CBC64Context *pContext,
        UINT32 *pData,
        unsigned NumDWORDBlocks)
{

    UINT32 Checksum;
    register UINT32 Datum;
    register UINT32 Key1;
    register UINT32 Key2;
    UINT32 numQuadBlocks;

    const UINT32 CBC_AB[2] = { CBC_b, CBC_bXORa };
    const UINT32 CBC_CD[2] = { CBC_d, CBC_dXORc };

    if (NumDWORDBlocks != 0) {
        Checksum = pContext->Checksum;
        Datum = pContext->Datum;
        Key1 = pContext->Key1;
        Key2 = pContext->Key2;

        INT iterStart = NumDWORDBlocks % 8;

         if (iterStart) {
            NumDWORDBlocks -= iterStart;
            do {
                        Checksum += *pData;
                        Datum = CBC_RandomOddAlpha * (*pData + Datum)+\

      CBC_RandomBeta;
                        Key1 ^= Datum;
                        Key1 = (Key1 << 1)^(CBC_CD[(Key1&0x80000000) >>
31]);
                        Key2 ^= Datum;
                        Key2 = (Key2 << 1)^(CBC_AB[(Key2&0x80000000) >>
31]);
                  pData++;
                  iterStart--;
            } while (iterStart);
        }

      __m128i xmmRawData;
      __m128i xmmChecksum;
      __m128i xmmDatum;
      __m128i xmmCBC_RandomAlpha;
      __m128i xmmCBC_RandomBeta;
      __m128i xmm1;
      __m128i xmmKey1;
      __m128i xmmKey2;

      xmmChecksum = _mm_set_epi32((int)Checksum, (int)Checksum,
(int)Checksum, (int)Checksum);
      xmmCBC_RandomAlpha = _mm_set_epi32((int)CBC_RandomOddAlpha,
       (int)CBC_RandomOddAlpha, (int)CBC_RandomOddAlpha,
(int)CBC_RandomOddAlpha);
      xmmCBC_RandomBeta = _mm_set_epi32((int)CBC_RandomBeta,
(int)CBC_RandomBeta, (int)CBC_RandomBeta, (int)CBC_RandomBeta);
      xmmDatum = _mm_setzero_si128();

      xmmKey1 = _mm_set_epi32((int)Key1, (int)Key1+CBC_RandomOddAlpha,
(int)Key1+CBC_RandomBeta, (int)Key1-CBC_RandomBeta);
      xmmKey2 = _mm_set_epi32((int)Key2, (int)Key2+CBC_RandomOddAlpha,
(int)Key2+CBC_RandomBeta, (int)Key2-CBC_RandomBeta);

      PUINT32 pDataEnd = pData +numQuadBlocks*8;

      while (pData < pDataEnd) {
            xmmRawData = _mm_load_si128((__m128i*)pData);
            pData += 4;

            xmmChecksum = _mm_add_epi32(xmmRawData, xmmChecksum);
[0057]      xmm1 = _mm_add_epi32(xmmRawData, xmmDatum);

            xmm1 = _mm_madd_epi16(xmm1, xmmCBC_RandomAlpha);
            xmmDatum = _mm_add_epi32(xmm1, xmmCBC_RandomBeta);
            xmmKey1 = _mm_xor_si128(xmmKey1, xmmDatum);
            xmmKey2 = _mm_xor_si128(xmmKey2, xmmDatum);
            xmmRawData = _mm_load_si128((__m128i*)pData);
            pData += 4;

            xmmChecksum = _mm_add_epi32(xmmRawData, xmmChecksum);

            xmm1 = _mm_add_epi32(xmmRawData, xmmDatum);
            xmm1 = _mm_madd_epi16(xmm1, xmmCBC_RandomAlpha);
            xmmDatum = _mm_add_epi32(xmm1, xmmCBC_RandomBeta);

            xmmKey1 = _mm_xor_si128(xmmKey1, xmmDatum);
            xmmKey2 = _mm_xor_si128(xmmKey2, xmmDatum);

            }

      PUINT32 pQuadKey1 = (PUINT32)&xmmKey1;
      PUINT32 pQuadKey2 = (PUINT32)&xmmKey2;
      PUINT32 pQuadDatum = (PUINT32)&xmmDatum;

      #define BLEND_QUAD(quad) (quad[0] ^ quad[1] ^ quad[2] ^ quad[3])

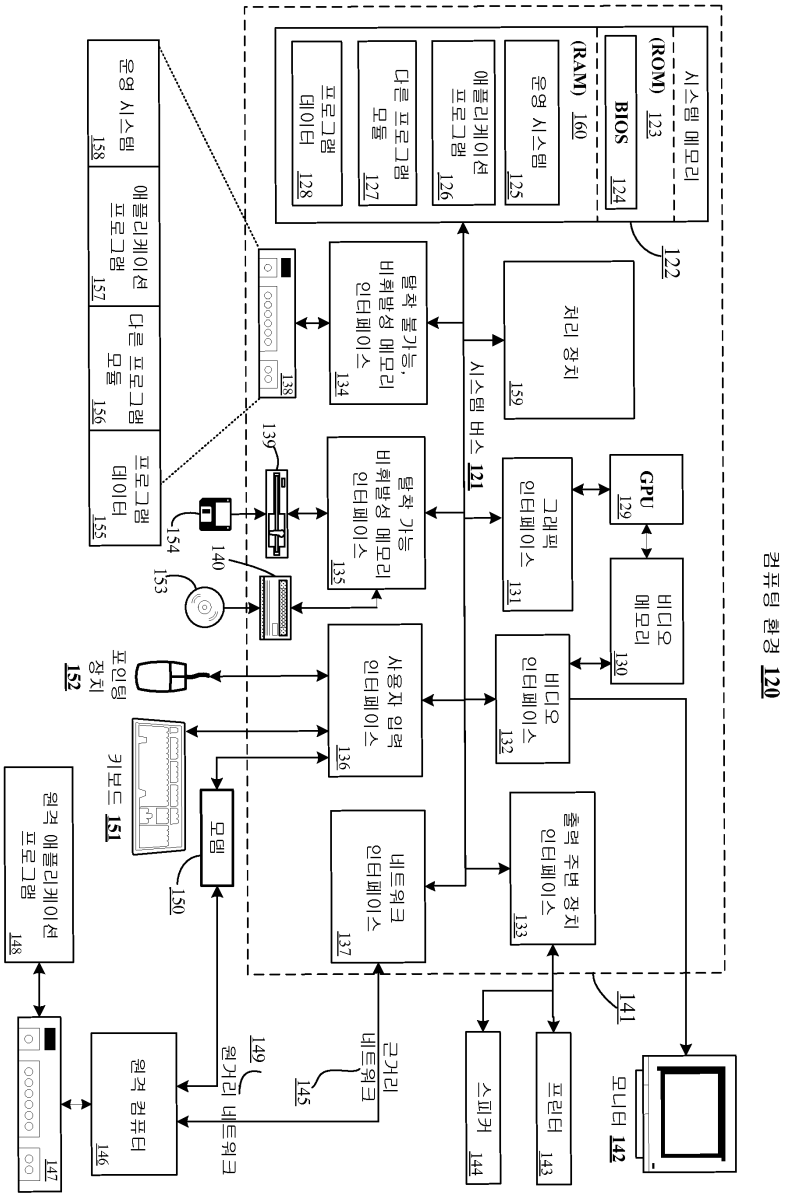
      pContext->Checksum = Checksum;
      pContext->Datum = Datum ^ BLEND_QUAD(pQuadDatum);
      pContext->Key1 = Key1 ^ BLEND_QUAD(pQuadKey1);
      pContext->Key2 = Key2 ^ BLEND_QUAD(pQuadKey2);
      }
}
```

[0058] 결론

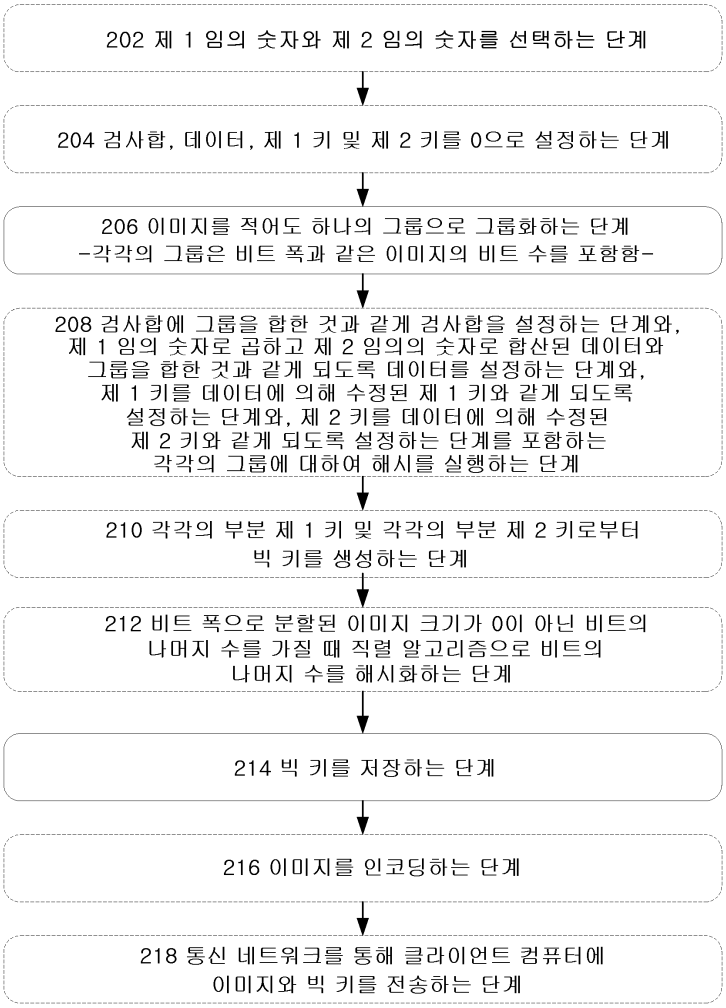
[0059] 본 발명은 여러 도면에 도시된 바와 같이 바람직한 측면을 참조하여 설명되어 있으나, 본 발명을 개시 내용을 벗어나지 않으면서 다른 유사한 측면이 이용될 수 있고 본 발명의 개시 내용과 동일한 기능을 수행하는 것으로 설명된 측면에 대한 수정 및 추가가 이루어질 수 있다는 것을 이해할 것이다. 그러므로 본 발명은 임의의 단일 측면으로 제한되지 않아야 하지만, 첨부된 청구항에 따른 범위 및 범주 내에는 속하는 것으로 간주되어야 한다. 예를 들면, 본 명세서에 설명된 여러 절차는 하드웨어 또는 소프트웨어로 구현되거나, 그 조합으로 구현될 수 있다. 따라서 본 발명의 실시예의 방법 및 장치 또는 그에 대한 소정의 측면 또는 부분은 플로피 디스크, CD-ROM, 하드 드라이브 또는 임의의 다른 머신 판독 가능 저장 매체 등과 같은 실체적 매체 내에 구현되는 프로그램 코드(즉, 인스트럭션)의 형태를 가질 수 있다. 프로그램 코드가 컴퓨터 등과 같은 머신에 로딩되고 머신에 의해 실행될 때, 머신은 본 명세서에 개시된 실시예를 실행하도록 구성된 장치가 된다. 당업자라면 본 명세서에 명확히 제시된 특정한 구현에 추가하여, 본 명세서에 개시된 설명을 고려함으로써 다른 측면 및 구현을 명확히 인식할 것이다. 설명 및 제시된 구현은 오로지 예시로만 고려되도록 의도되었다.

도면

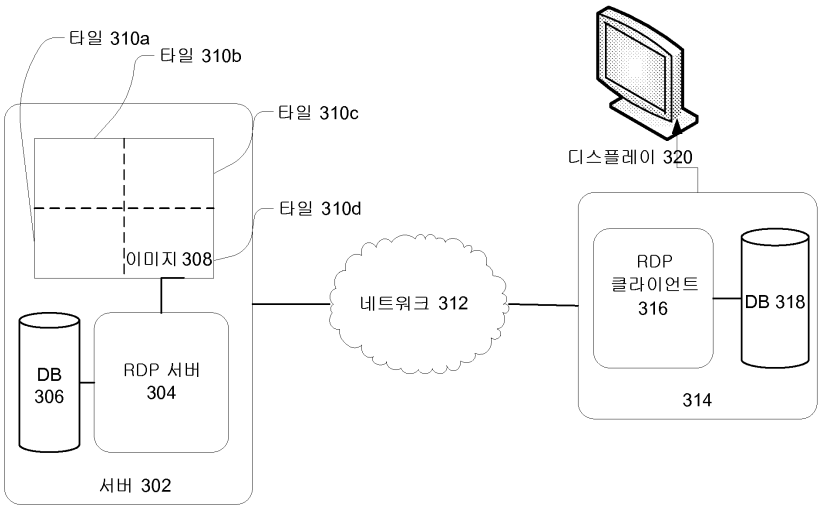
도면1



도면2



도면3



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 제19항

【변경전】

상기 비 키

【변경후】

상기 빅 키