US 20120331377A1

(19) United States
(12) Patent Application Publication
Howell et al.
(10) Pub. No.: US 2012/0331377 A1
(43) Pub. Date: Dec. 27, 2012

(54) CONTENT RENDERING ON A COMPUTER

(75) Inventors: **Gareth Alan Howell**, Bothell, WA (US); **Matthew James Ruhlen**, Redmond, WA (US); **Brendyn O'Dell-Alexander**, Seattle, WA (US); **Nicholas Allen Robarge**, Redmond, WA (US); **Zhenjun Zhu**, Redmond, WA (US); **Sugandha SudeshKumar Kapoor**, Sammamish, WA (US); **Kiran Akella Venkata**, Bellevue, WA (US); **Juraj Gottweis**, Bellevue, WA (US)

(73) Assignee: **MICROSOFT CORPORATION**, Redmond, WA (US)

(21) Appl. No.: **13/606,704**

(22) Filed: **Sep. 7, 2012**

Related U.S. Application Data

(63) Continuation of application No. 12/397,351, filed on Mar. 4, 2009.

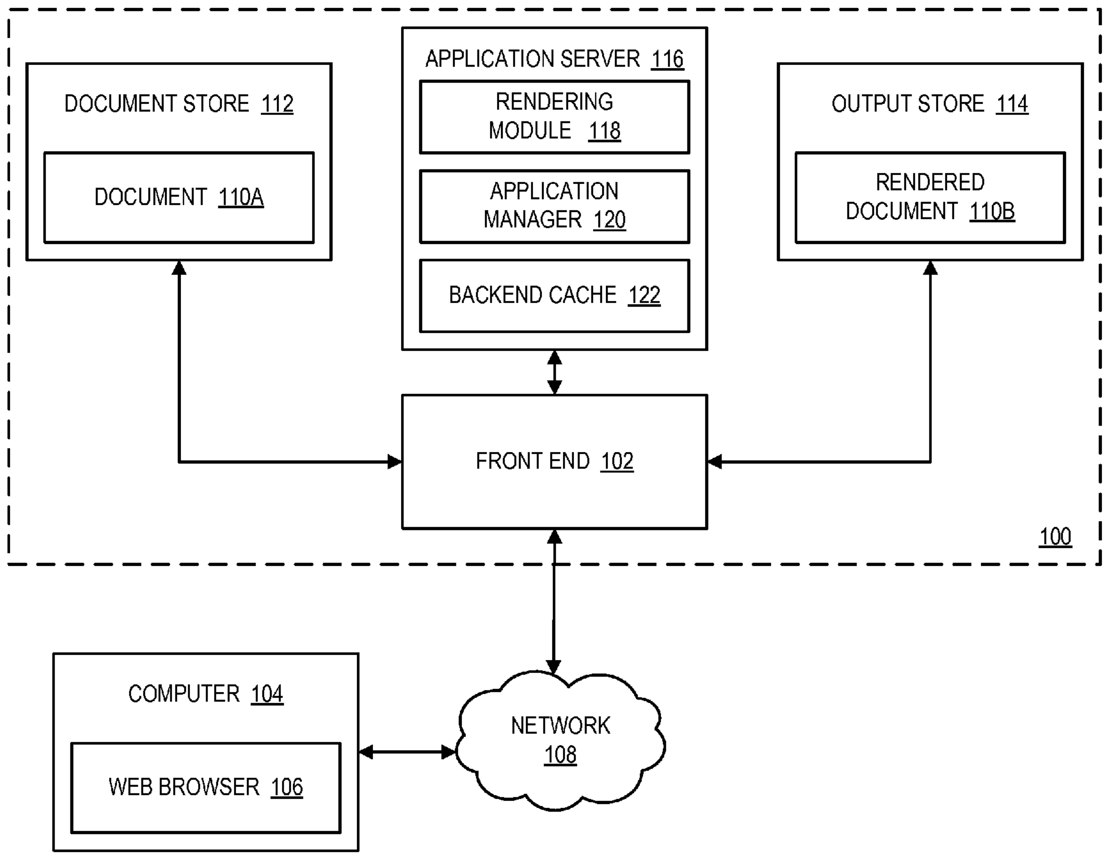
Publication Classification

(51) **Int. Cl.**
G06F 3/01 (2006.01)
G06F 17/00 (2006.01)

(52) **U.S. Cl.** **715/234**; 715/760

(57) **ABSTRACT**

Portions of content are transformed into portions of rendered content. While the portions of the content are being transformed into portions of the rendered content, each discrete portion of the rendered content can be provided to the application program after that portion is completed.



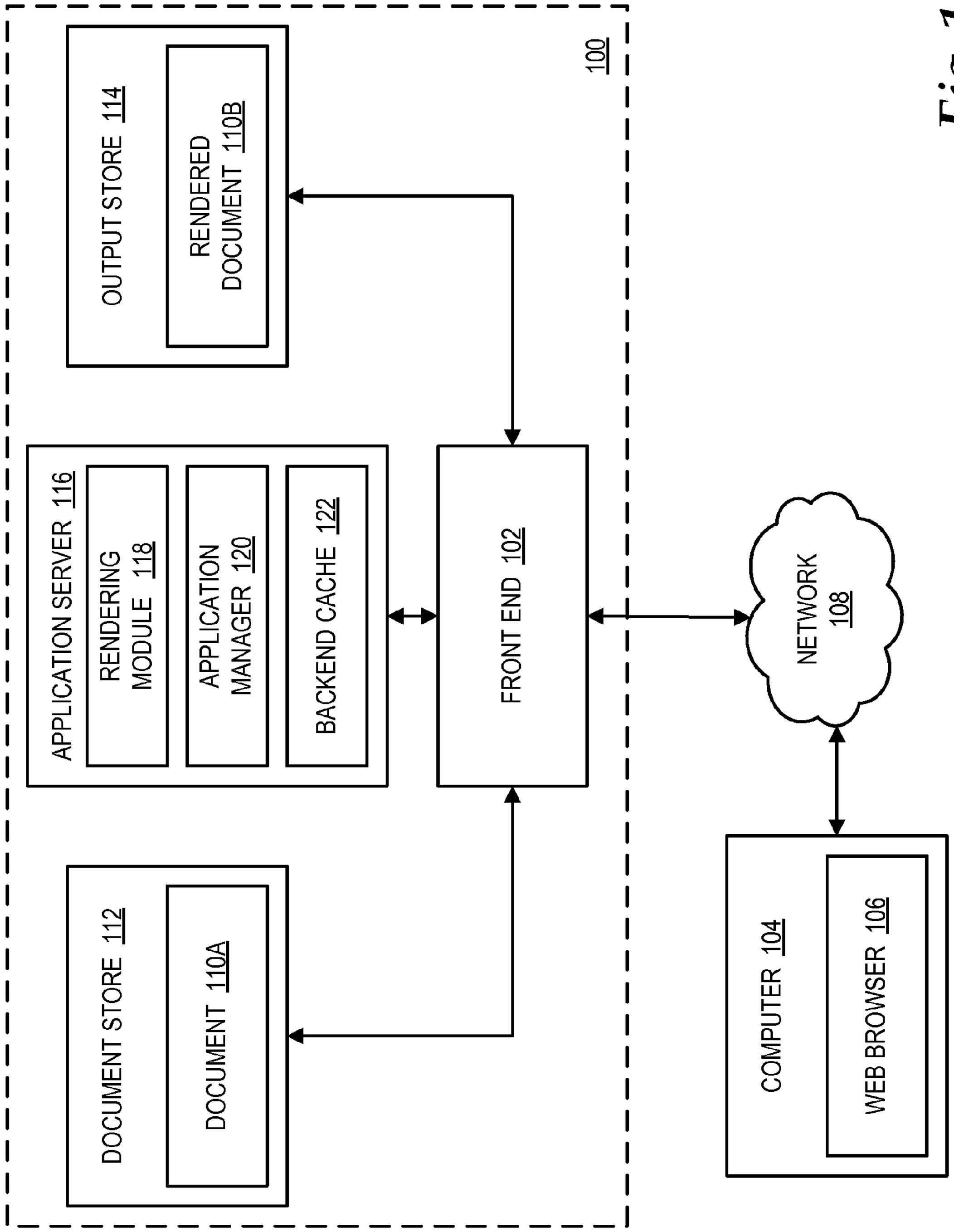

OUTPUT STORE 114
RENDERED DOCUMENT 110B
100
APPLICATION SERVER 116
RENDERING MODULE 118
APPLICATION MANAGER 120
BACKEND CACHE 122
FRONT END 102
NETWORK 108
DOCUMENT STORE 112
DOCUMENT 110A
COMPUTER 104
WEB BROWSER 106

Fig. 1

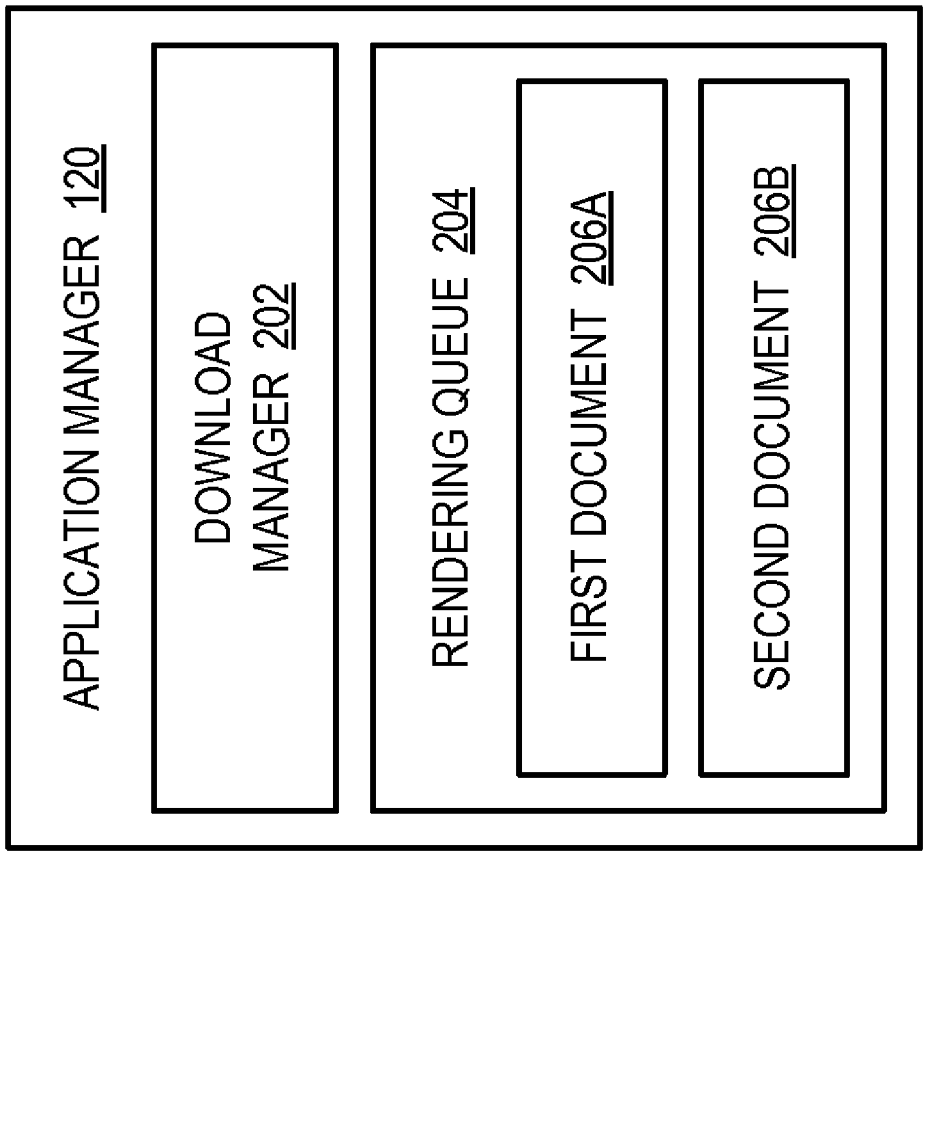

APPLICATION MANAGER 120
DOWNLOAD MANAGER 202
RENDERING QUEUE 204
FIRST DOCUMENT 206A
SECOND DOCUMENT 206B

Fig. 2

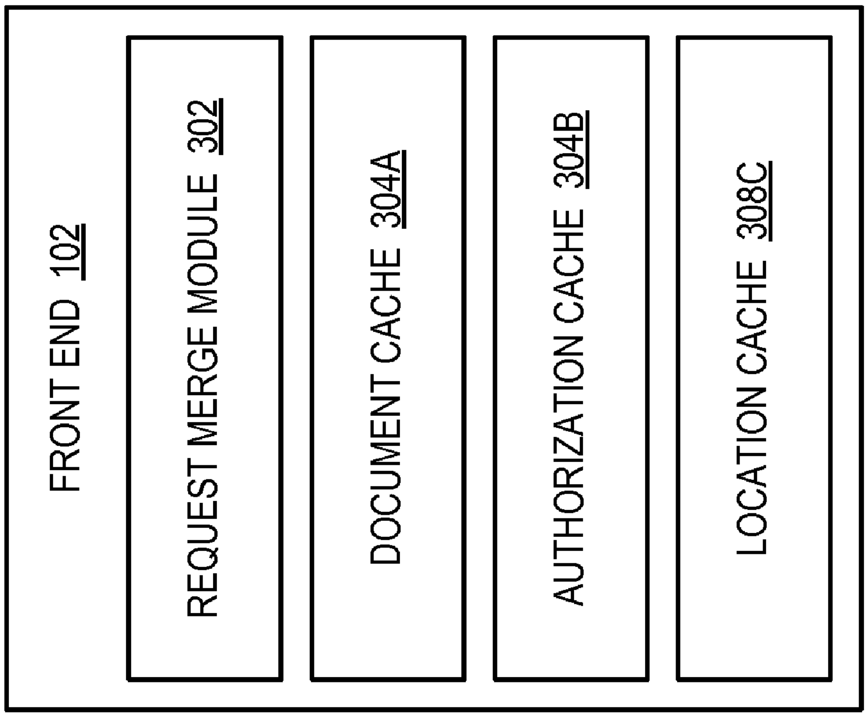

FRONT END 102
REQUEST MERGE MODULE 302
DOCUMENT CACHE 304A
AUTHORIZATION CACHE 304B
LOCATION CACHE 308C

Fig. 3

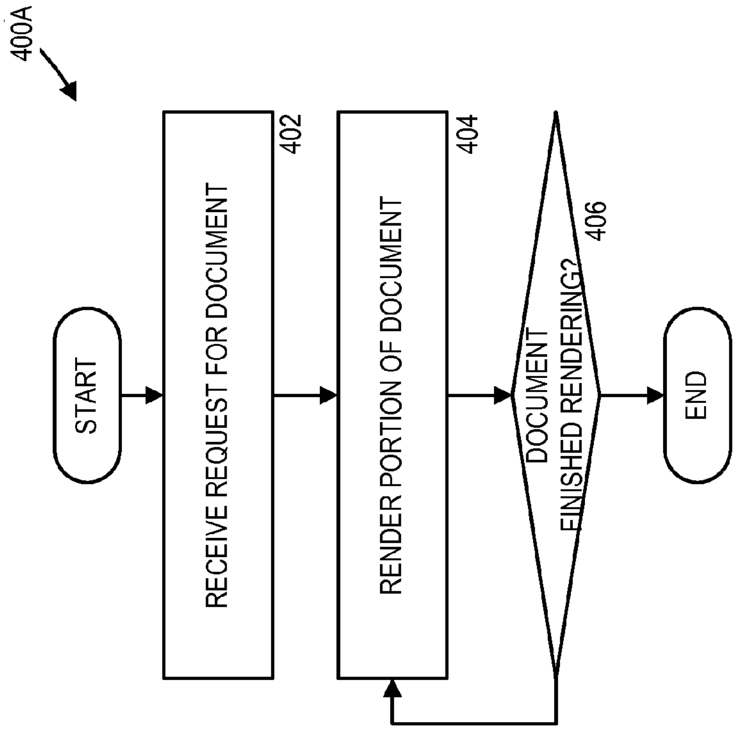

400A
START
RECEIVE REQUEST FOR DOCUMENT
402
RENDER PORTION OF DOCUMENT
404
DOCUMENT FINISHED RENDERING?
406
END

Fig. 4A

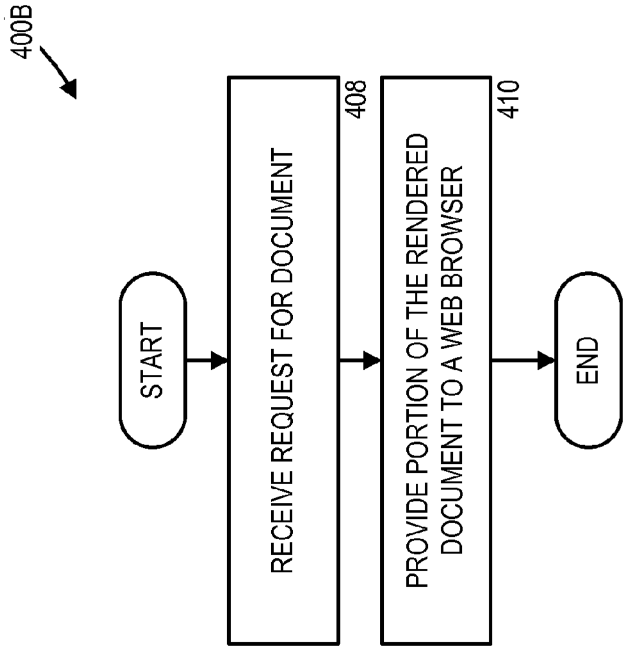

400B
START
RECEIVE REQUEST FOR DOCUMENT
408
PROVIDE PORTION OF THE RENDERED DOCUMENT TO A WEB BROWSER
410
END

Fig. 4B

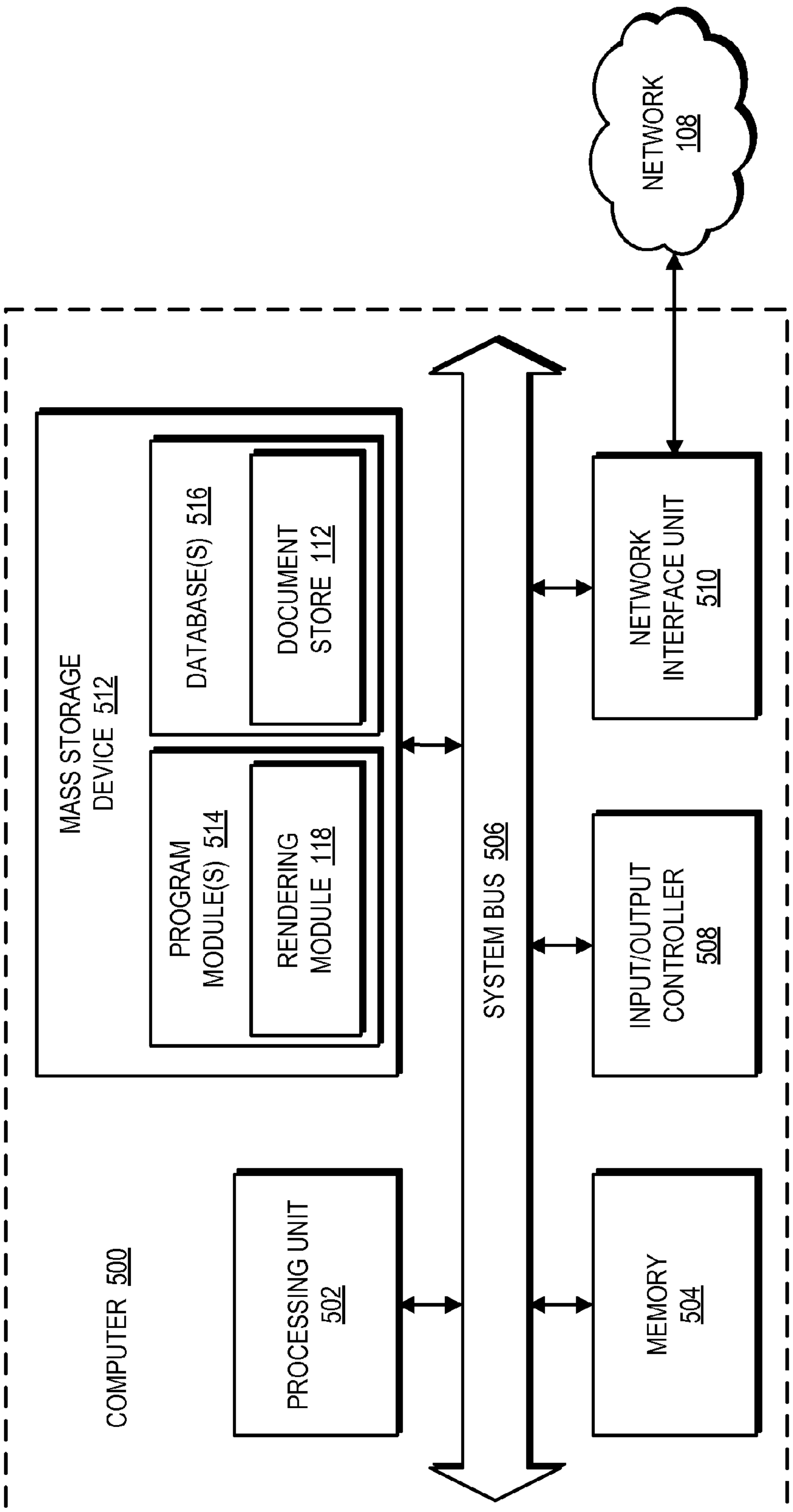

COMPUTER 500
PROCESSING UNIT 502
MASS STORAGE DEVICE 512
PROGRAM MODULE(S) 514
RENDERING MODULE 118
DATABASE(S) 516
DOCUMENT STORE 112
SYSTEM BUS 506
MEMORY 504
INPUT/OUTPUT CONTROLLER 508
NETWORK INTERFACE UNIT 510
NETWORK 108

Fig. 5

CONTENT RENDERING ON A COMPUTER

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of co-pending U.S. patent application Ser. No. 12/397,351 entitled "Content Rendering on a Computer," filed Mar. 4, 2009, which is expressly incorporated by reference herein.

BACKGROUND

[0002] Productivity application programs, such as word processing applications, spreadsheet applications, and presentation applications, enable users to create different types of documents. These documents may include data that is native to a particular document type and even to a particular application program. In order to view this native data, a user may view the document through the productivity application program that created the document. For example, a user may rely on a particular word processing application to view a word processing document that was created by the word processing application.

[0003] A problem can arise when documents are shared across the World Wide Web (hereinafter referred to as "web") through web browsers. In particular, typical web browsers may not be capable of reading a given document in its native format. In order to address this problem, a web server can be implemented to execute code that is capable of rendering a document from its native format into a web-compatible format that can be viewed through the web browser.

[0004] Conventional implementations of such web servers render a document in its entirety prior to providing the rendered version to the web browser. This is sometimes referred to as document pre-rendering. During pre-rendering, a user who requests to view a given document may be forced to wait a significant amount of time before the document has been completely rendered. This delay can cause substantial frustration and inconvenience for the user.

[0005] It is with respect to these considerations and others that the disclosure made herein is presented.

SUMMARY

[0006] Technologies are described herein for rendering content, such as documents. An application server is provided for rendering documents and other content from a native document format into a web-compatible format capable of being viewed through a web browser or other suitable application. The application server includes a rendering module, which is operative to incrementally render documents. Through incremental rendering, the application server can provide portions of rendered documents to the web browser while the native documents are still being rendered. Other example implementations of the technologies described herein include converting a document from one format into another format (e.g., transforming a document from a binary file format into an Office Open XML format, a Portable Document Format ("PDF"), and the like).

[0007] A download manager is also provided in order to increase the throughput of the rendering module. While the rendering module is rendering one document, the download manager may begin retrieving other documents to be rendered at a later time. These other documents may be downloaded from a storage server and stored in a rendering queue on the application server. Once the rendering module becomes available, the rendering module can immediately begin rendering these other documents directly from the rendering queue without any downtime while waiting for documents to be retrieved.

[0008] One or more caches are also provided in order to reduce the amount of work utilized to retrieve documents and other content. Examples of caches include a document cache, an authorization cache, and a location cache. The document cache may be operative to store rendered documents in the front end, thereby eliminating the need to retrieve the rendered documents from other storage units in the back end. The authorization cache may be operative to store authorization and authentication data, thereby eliminating the need to reauthorize or re-authenticate a user who later accesses the same document. The location cache may be operative to store the location of a rendered document. Through the use of the location cache, the rendered document can be efficiently retrieved from the relevant storage unit without the need to query multiple storage units.

[0009] According to one embodiment, a method is provided herein for rendering content for viewing through a web browser operating on a computer. Portions of the content are transformed into portions of rendered content. While the portions of the content are being transformed into portions of the rendered content, at least one of the portions of the rendered content can be provided to an application program.

[0010] It should be appreciated that the above-described subject matter may also be implemented as a computer-controlled apparatus, a computer process, a computing system, or as an article of manufacture such as a computer-readable medium. These and various other features will be apparent from a reading of the following Detailed Description and a review of the associated drawings.

[0011] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended that this Summary be used to limit the scope of the claimed subject matter. Furthermore, the claimed subject matter is not limited to implementations that solve any or all of the disadvantages noted in any part of this disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] FIG. **1** is a block diagram showing an illustrative system architecture operative to render documents for viewing through a web browser, in accordance with embodiments;

[0013] FIG. **2** is a block diagram showing an illustrative implementation of an application manager from the system architecture of FIG. **1**, in accordance with embodiments;

[0014] FIG. **3** is a block diagram showing an illustrative implementation of a front end from the system architecture of FIG. **1**, in accordance with embodiments;

[0015] FIG. **4**A is a flow diagram showing an illustrative process for rendering content for viewing through a web browser, in accordance with embodiments;

[0016] FIG. **4**B is a flow diagram showing an illustrative process for fetching a portion of the rendered content, in accordance with embodiments; and

[0017] FIG. **5** is a computer architecture diagram showing an illustrative computer hardware architecture for a computing system capable of implementing the embodiments presented herein.

DETAILED DESCRIPTION

[0018] The following detailed description is directed to technologies for rendering content, such as documents. Through the utilization of the technologies and concepts presented herein, an application server is provided that is operative to incrementally render a document or other content from its native format into another format, such as a web-compatible format. In contrast to pre-rendering, as content is being incrementally rendered, each discrete portion of the document or other content that has been rendered can be made available to the user. One or more caches may also be provided in order to reduce the amount of work utilized to retrieve rendered content.

[0019] Embodiments described herein may refer to pages and slides as discrete portions of documents and rendered documents. However, it should be appreciated that pages and slides are merely examples of discrete portions and are not intended to be limiting. Other kinds of discrete portions, as contemplated by those skilled in the art, may be similarly implemented.

[0020] Although not so limited, embodiments described herein primarily refer to documents and document rendering. However, it should be appreciated that the embodiments described herein may be similarly applied to any suitable content which can be rendered from a native format into another format. Such content may include any multimedia, such as text, images, audio, video, and combinations thereof. Further, embodiments described herein primarily refer to the rendered format as a web-compatible format capable of being viewed through a web browser. However, it should be appreciated that the rendered format may be other suitable formats as contemplated by those skilled in the art.

[0021] While the subject matter described herein is presented in the general context of program modules that execute in conjunction with the execution of an operating system and application programs on a computer system, those skilled in the art will recognize that other implementations may be performed in combination with other types of program modules. Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the subject matter described herein may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like.

[0022] In the following detailed description, references are made to the accompanying drawings that form a part hereof, and which are showing by way of illustration specific embodiments or examples. Referring now to the drawings, in which like numerals represent like elements through the several figures, concepts and technologies for providing on-demand content viewing will be described. In particular, FIG. **1** is a block diagram showing a system architecture **100** operative to render documents and to retrieve rendered documents according to embodiments. The architecture **100** includes a front end **102** that is operative to communicate with a computer **104**. The front end **102** may receive and process requests from a web browser **106** over a network **108**, such as the Internet. For example, the front end **102** may receive a Hypertext Transfer Protocol ("HTTP") message with a request for a specific document, such as document **110**A, stored in a document store **112**. It should be appreciated that the components of the architecture **100** may reside on the same server, according to further embodiments. Splitting the architecture **100** into the front ends and application servers is merely an implementation design choice and is not intended to be limiting.

[0023] In response to receiving the HTTP request, the web front end **102** may attempt to retrieve, from an output store **114**, a web-compatible rendered document **110**B corresponding to the requested document **110**A. For example, the rendered document **110**B may be stored in the output store **114** if the document **110**A was recently rendered. If the output store **114** contains the rendered document **110**B corresponding to the document **110**A, then the output store **114** may return the rendered document **110**B to the front end **102**.

[0024] If the output store **114** does not contain the rendered document **110**B, then the application server **116**, and more specifically the rendering module **118**, may retrieve the document **110**A from the document store **112**. According to embodiments, a rendering module **118** in the application server **116** is operative to render (i.e., convert, transform) the document **110**A into the rendered document **110**B. The rendering module **118** may generate the rendered document **110**B in any format capable of being displayed by a suitable web browser. Example formats may include an image format (e.g., Portable Network Graphics ("PNG"), Joint Photographic Experts Group ("JPEG"), etc.), MICROSOFT SILVERLIGHT, ADOBE FLASH, and the like. The rendering module **118** may also generate the rendered document **110**B into a web-ready rendition that may include Extensible Markup Language ("XML"), scripts, Hypertext Markup Language ("HTML"), images, and/or the like, such that individual data types in the web-ready rendition can be understood collectively by the web browser. Each of these data types may be made available to the user upon being generated by the incremental rendering process described herein. The rendering module **118** may also be operative to concurrently render multiple documents. Upon generating the rendered document **110**B, the application server **116** may store the rendered document **110**B in the output store **114** and return the rendered document **110**B to the front end **102**.

[0025] Upon receiving the rendered document **110**B, the front end **102** may respond to the HTTP request by providing the rendered document **110**B to the web browser **106** via the HTTP protocol. The web browser **106** can then display the rendered document **110**B, which is a full-fidelity representation of the corresponding document **110**A. The full-fidelity representation contains the same formatting (e.g., layout, resolution, content, etc.) found in the original representation. Thus, a user can expect the same experience viewing a given document whether the document is viewed through a productivity application program or through a web browser, such as the web browser **106**. The full-fidelity representation can be contrasted against a lower-fidelity representation in which, for example, the layout may be changed or the resolution and content may be reduced. In other embodiments, the rendered document **110**B may be utilized for non-viewing scenarios, such as converting binary files to another document type for editing. For example, a MICROSOFT OFFICE binary file may be rendered into an OPEN OFFICE EXTENSIBLE MARKUP LANGUAGE ("OOXML") file.

[0026] It should be appreciated that the components and the configuration of the components within the architecture **100** of FIG. **1** are merely illustrative and are not intended to be limiting. In other embodiments, the architecture **100** may include additional front ends and/or additional application servers. Further, multiple front ends and/or multiple application servers may reside on the same server. For example, additional front ends may be utilized to handle a greater quantity of requests from web browsers. In one embodiment, the additional application servers are deployed to handle the rendering of heavier loads. In another embodiment, additional application servers are deployed to render different types of documents. For example, one application server may be dedicated to rendering word processing documents, while another application server may be dedicated to rendering presentations. In yet another embodiment, additional application servers are deployed to render different elements of a given document. The rendered elements can then be combined to create the rendered document. For example, one application server may render video in a presentation, while another application server may render the text in the presentation. In this case, the front end **102** may be operative to combine the results of the two application servers to form the rendered presentation.

[0027] In some embodiments, in response to a single request to render a document in a given format, the front end **102** may direct the application server **116** to render the document into multiple formats in order to support multiple uses across multiple device types and web client configurations. For example, the web browser **106** may request from the front end **102** a given document rendered in ADOBE FLASH. The front end **102** may determine, through a user profile associated with the user operating the web browser **106**, that the user frequently accesses documents in ADOBE FLASH, MICROSOFT SILVERLIGHT, Portable Network Graphics ("PNG"), mobile, and full versions. The front end **102** may also determine, through the user profile that the user frequently accesses documents in a web-ready rendition including a combination of XML, scripts, HTML, images, and/or the like. In this case, the front end **102** may fulfill the request by rendering the document in one or more of these versions. Other suitable data may be similarly utilized to determine whether the front end **102** directs the application server **116** to render a document into multiple formats in response to a single render request. By proactively rendering documents into multiple formats in response to a single request rather than rendering the document in response to specific requests for the additional formats, the architecture **100** can more expeditiously provide the rendered documents.

[0028] According to embodiments, the application server **116** further includes an application manager **120**. The application manager **120** is operative to manage incremental rendering of the document **110**A into the rendered document **110**B. As used herein, "incremental rendering" refers to providing portions of the rendered document **110**B as the document **110**A is being rendered by the rendering module **118**. For example, the rendered first page of a word processing document or a rendered first slide of a presentation may be provided to the front end **102**, which can then provide the rendered first page or the rendered first slide to the web browser **106** for display. Other embodiments may render any particular page or slide in the document. Other embodiments may also render other portions of content besides pages or slides. As the remainder of the word processing document or presentation is being rendered by the rendering module **118**, the application manager **120** may provide the additional rendered pages and rendered slides to the web browser **106**.

[0029] In order to provide incremental rendering, the application manager **120** may make available portions of the rendered document **110**B as the document **110**A is being rendered by the rendering module **118**. The front end **102** can then retrieve these portions of the rendered document **110**B, as necessary, from the application server **116**. In one embodiment, the portions of the rendered document **110**B are stored in a backend cache **122** in the application server **116**.

[0030] During incremental rendering, the pages or slides of a document may be rendered in any predefined order. The rendered pages or slides are then made available to the front end **102** in the same order as they were rendered. In one embodiment, the documents are rendered from the beginning of the document to the end of the document.

[0031] In contrast to conventional pre-rendering where a user must wait until the entire document is rendered, incremental rendering reduces the amount of rendering time before a user can view a portion, such as the beginning or some other portion, of the document. For example, in conventional pre-rendering, a user desiring to view a first page of a word processing document or a first slide of a presentation would need to wait until the entire word processing document or presentation is rendered. In an illustrative implementation of incremental rendering, the user can more quickly view the first page or the first slide prior to the remainder of the word processing document or the presentation being rendered.

[0032] In some embodiments, the user may request a page N or a slide N of a document where the variable N represents any page or slide within the document. The request may occur prior to or while the document is being rendered. Upon receiving the request for the page N or slide N, the application manager **120** may direct the rendering module **118** to render the page N or the slide N by seeking the particular page or slide within the document. The rendered page N or slide N can then be made available to the front end **102**. Upon making available the rendered page N or slide N, the application manager **120** may proceed with rendering the remainder of the document.

[0033] In some embodiments, the rendering module **118** is operative to incrementally render a document even when the document has yet to be fully downloaded. That is, the rendering module **118** may begin rendering the document while it is being downloaded from the document store **112**, rather than wait until the entire file has been downloaded. This approach to rendering a document before it has fully downloaded may reduce the amount of time it takes for the rendering module **118** to render a document, especially if the document is relatively large.

[0034] Turning now to FIG. **2**, additional details regarding the application manager **120** will be described. In particular, FIG. **2** is a block diagram showing the application manager **120** according to embodiments. The application manager **120** includes a download manager **202** and a rendering queue **204**. According to embodiments, the download manager **202** is operative to download documents from the document store **112** prior to the rendering module **118** rendering the documents. The download manager **202** may then store the downloaded documents in the rendering queue **204** until the rendering module **118** is available to render the documents. It should be appreciated that the rendering queue **204** may operate without the download manager **202**. However, the download manager **202** may be utilized to speed processing through the rendering queue **204**. Further, even when the download manager **202** is present, the rendering queue **204**

may operate independently of the download manager **202** to handle instances when the application manager **120** is overloaded.

[0035] In an illustrative example, if a request is made to render a first document **206**A and a second document **206**B, the download manager **202** may download the first document **206**A from the document store **112**, and the rendering module **118** may begin rendering the first document **206**A. As the rendering module **118** renders the first document **206**A, the download manager **202** may concurrently download the second document **206**B from the document store **112** and store the second document **206**B in the rendering queue **204**. Upon rendering the first document **206**A, the rendering module **118** can then immediately proceed to rendering the second document **206**B from the rendering queue **204** without waiting for the second document **206**B to be downloaded. In this way, throughput of the rendering module **118** is improved by eliminating any idle time found in conventional implementations where the rendering module **118** would wait for the second document **206**B to download.

[0036] It should be appreciated that the rendering queue **204** may be configured to store any number of documents, as contemplated by those skilled in the art. In some embodiments, the download manager **202** may be configured by a user to adjust the number of documents being downloaded simultaneously and the maximum number of documents stored in the rendering queue **204** that are awaiting rendering. In this way, the user can ensure that the number of documents being downloaded and/or stored at any given time does not place an excessive burden on system resources.

[0037] Turning now to FIG. **3**, additional details regarding the front end **102** will be described. In particular, FIG. **3** is a block diagram showing the front end **102** according to embodiments. The front end **102** includes a request merge module **302** and several caches **304**, including a document cache **304**A, an authorization cache **304**B, and a location cache **304**C. According to embodiments, the request merge module **302** maintains tracking data identifying a particular application server that is currently rendering a given document. The request merge module **302** may utilize this tracking data to direct additional requests for the same document to the same application server. By directing additional requests for the same document to the same application server, redundant work between application servers is reduced, and other application servers are free to render other documents.

[0038] As previously described, the system architecture **100** of FIG. **1** may include multiple front ends and multiple application servers. These front ends may or may not be capable of communicating with each other. In the case where front ends do not communicate with each other, the request merge module **302** may be configured to perform consistent hashing in order to identify a particular application server for a particular document request. In consistent hashing, one or more aspects of a given document request may be analyzed in order to map these aspects to a particular application server. In this way, separate front ends can direct multiple requests for the same document to the same application server, even when the front ends cannot communicate with each other. An example of an aspect of a document request is the name of the document being requested. It should be appreciated that consistent hashing in this case can significantly increase the likelihood that documents are not rendered multiple times on different application servers (e.g., multiple front ends could receive requests for the same document). Without consistent hashing, many application servers may receive these requests and process the same document. Other hashing techniques capable of mapping document requests to application servers may be similarly utilized.

[0039] The caches **304** may improve performance of the front end **102** by reducing the need for the front end **102** to access other parts of the system architecture **100**, such as the output store **114** and the application server **116**. According to embodiments, the document cache **304**A is operative to cache at least portions of rendered documents according to popularity. For example, documents that are frequently requested can be retrieved directly from the document cache **304**A instead of accessing the output store **114** or the application server **116**. Any of a variety of factors and methodologies may be utilized for determining the popularity of documents, as contemplated by those skilled in the art.

[0040] According to embodiments, the authorization cache **304**B is operative to store authorization or authentication information associated with users. Access to the documents and the rendered documents may be restricted based on an authorization process whereby a user enters identification information, such as a username and password. For example, when a user attempts to access a particular document, the front end **102** may access an authorization module (not shown), which requests information, such as a login identifier and password, identifying the user. The authorization module may then determine whether the user has permission to access the document. The authorization module may be configured to permit access to the document only when the authorization module verifies that the user has permission.

[0041] When the authorization module successfully verifies the user, the authorization cache **304**B may store state information indicating that the user has been verified. The front end **102** may utilize this state information to verify that the user has or does not have permission to access the document without the need for accessing the authorization module. In this way, the front end **102** is not required to send credentials to an remote authorization service, thereby optimizing performance of the front end **102**. Further, the user does not have to re-enter identification information for additional requests to access the same document.

[0042] The state information stored in the authorization cache **304**B may be particularly useful when a user requests only a portion of a given document at a time. For example, a user may initially request only a first page of a multi-page document. Upon submitting the request for the first page of the document, the user may be required to enter identification information (i.e., credentials) in order to view the first page. If the user subsequently requests additional pages of the document, the front end **102** can expeditiously authorize the user based on the credentials stored in the authorization cache **304**B. Thus, each authentication of the user does not necessarily result in the user being prompted for credentials. Further, the need to re-query other servers to authenticate and authorize the user can be eliminated. The authorization cache **304**B may be configured to store the state information for any amount of time after which the state information may be deleted.

[0043] According to embodiments, the location cache **304**C, which may also be referred to as a "dictionary", is operative to store the location of a given rendered document. For example, the location cache **304**C may indicate the particular storage unit that stores a particular rendered document. Examples of storage units may include the output store **114**

and the backend cache **122**. The storage units may be part of the system architecture **100** or separate from the system architecture **100**. By storing the location of rendered documents in the location cache **304**C, the front end **102** can direct requests for a rendered document to the specific storage unit, as opposed to individually querying each storage unit and causing additional loads on the storage units.

[0044] Referring now to FIGS. **4**A and **4**B, additional details will be provided regarding the embodiments presented herein for rendering documents for viewing through a web browser. In particular, FIG. **4**A is a flow diagram illustrating one method for rendering content for viewing through a web browser operating on a computer. FIG. **4**B is a flow diagram illustrating one method for fetching a rendered portion of the content. The method of FIG. **4**B may be performed concurrently or after the method of FIG. **4**A is performed. It should be appreciated that the logical operations described herein are implemented (1) as a sequence of computer implemented acts or program modules running on a computing system and/or (2) as interconnected machine logic circuits or circuit modules within the computing system. The implementation is a matter of choice dependent on the performance and other requirements of the computing system. Accordingly, the logical operations described herein are referred to variously as states, operations, structural devices, acts, or modules. These operations, structural devices, acts, and modules may be implemented in software, in firmware, in special purpose digital logic, and any combination thereof. It should be appreciated that more or fewer operations may be performed than shown in the figures and described herein. These operations may also be performed in a different order than those described herein.

[0045] In FIG. **4**A, a routine **400** begins at operation **402**, where the front end **102** receives a request from the web browser **106** for a document, such as the document **110**A. The routine **400** then proceeds to operation **404**, where the front end **102** directs the rendering module **118** to begin incrementally rendering the document **110**A. In particular, the rendering module **118** may retrieve the document **110**A from the document store **112**. Upon retrieving the document **110**A from the document store **112**, the rendering module **118** may render the first page or the first slide of the document **110**A. In some embodiments, the request may specify a particular slide or page of the document **110**A. In this case, the rendering module **118** may render the particular page or the particular slide instead of the first page or the first slide. Upon generating the first page or the first slide of the rendered document **110**B, the routine proceeds to operation **406**.

[0046] At operation **406**, the rendering module **118** determines whether the document **110**A has been completely rendered. If the document **110**A has been completely rendered, then the routine **400**A terminates. If the document **110**A has not been completely rendered, the routine **400** proceeds back to operation **404** where the next page or next slide of the document **110**A is rendered. Operation **404** may be repeated as many times as necessary until the document **110**A has been completely rendered into the rendered document **110**B.

[0047] In FIG. **4**B, a routine **400**B begins at operation **408**, where the front end **102** receives a request from the web browser **106** for a document, such as the document **110**A. The routine **400**B then proceeds to operation **410**, where as pages or slides of the rendered document **110**B are rendered by the rendering module **118**, the application manager **120** may make available these pages or slides of the rendered document **110**B to the front end **102**. If a particular page or particular slide is requested, the application manager **120** may also make available that particular page or particular slide. The rendering module **118** may make available the pages or slides of the rendered document **110**B through the backend cache **122**. The front end **102** may then retrieve the pages or slides of the rendered document **110**B from the backend cache **122** and transmit the pages or slides of the rendered document **110**B to the web browser **106** for display.

[0048] It should be appreciated that the routine **400**B may be performed while the document **110**A is being incrementally rendered according to the routine **400**A. That is, while the rendering module **118** is rendering pages or slides of the document **110**A according to routine **400**A, the application manager **120** may fulfill requests for those rendered pages or slides according to routine **400**B. In this way, the front end **102** can continue to retrieve the additional pages or slides of the rendered document **110**B without waiting for the entire rendered document **110**B to be generated. Further, while the application manager **120** is fulfilling requests for pages or slides of the rendered document **110**B, the rendering module **118** may continue to render additional pages or slides from the document **110**A.

[0049] According to some embodiments, additional requests for other documents may be received by the front end **102**. Instead of waiting until the rendering module **118** has completed rendering the document **110**A into the rendered document **110**B, the download manager **202** may retrieve the other documents from the document store **112** and store the documents in the rendering queue **204** while the rendering module **118** is still rendering the document **110**A. Upon rendering the document **110**A into the rendered document **110**B, the rendering module **118** can begin rendering these other documents without waiting for them to be retrieved from document store **112**.

[0050] According to some embodiments, additional requests for the document **110**A may be directed to the same application server **116**. In this way, other application servers can be available to render other documents. Further, according to some embodiments, the rendering module **118** may render the document **110**A into multiple formats in response to a single request.

[0051] Referring now to FIG. **5**, an exemplary computer architecture diagram showing a computer **500** is illustrated. The computer **500** includes a processing unit **502** ("CPU"), a system memory **504**, and a system bus **506** that couples the memory **504** to the CPU **502**. The computer **500** further includes a mass storage device **512** for storing one or more program modules **514**, such as the rendering module **118**, and one or more databases **516**, such as the document store **112**. Other program modules **514** may include the web browser **106**, the download manager **202**, and the request merge module **302**. Other databases **516** may include the output store **114**, the rendering queue **204**, and the caches **304**. The mass storage device **512** is connected to the CPU **502** through a mass storage controller (not shown) connected to the bus **506**. The mass storage device **512** and its associated computer-readable media provide non-volatile storage for the computer **500**. Although the description of computer-readable media contained herein refers to a mass storage device, such as a hard disk or CD-ROM drive, it should be appreciated by those skilled in the art that computer-readable media can be any available computer storage media that can be accessed by the computer **500**.

[0052] By way of example, and not limitation, computer-readable media may include volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules, or other data. For example, computer-readable media includes, but is not limited to, RAM, ROM, EPROM, EEPROM, flash memory or other solid state memory technology, CD-ROM, digital versatile disks ("DVD"), HD-DVD, BLU-RAY, or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer **500**.

[0053] According to various embodiments, the computer **500** may operate in a networked environment using logical connections to remote computers through a network, such as the network **108**. The computer **500** may connect to the network **108** through a network interface unit **510** connected to the bus **506**. It should be appreciated that the network interface unit **510** may also be utilized to connect to other types of networks and remote computer systems. The computer **500** may also include an input/output controller **508** for receiving and processing input from a number of input devices (not shown), including a keyboard, a mouse, a microphone, and a game controller. Similarly, the input/output controller **508** may provide output to a display or other type of output device (not shown).

[0054] Based on the foregoing, it should be appreciated that technologies for rendering content, such as documents, for viewing through a web browser are presented herein. Although the subject matter presented herein has been described in language specific to computer structural features, methodological acts, and computer readable media, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features, acts, or media described herein. Rather, the specific features, acts and mediums are disclosed as example forms of implementing the claims.

[0055] The subject matter described above is provided by way of illustration only and should not be construed as limiting. Various modifications and changes may be made to the subject matter described herein without following the example embodiments and applications illustrated and described, and without departing from the true spirit and scope of the present invention, which is set forth in the following claims.

What is claimed is:

1. A method for rendering content on a computer, the method comprising:

receiving a request to view the content from a web browser;

in response to receiving the request to view the content, transforming a first portion of the content into a second portion of rendered content, wherein the rendered content is adapted to provide a representation of the content capable of being viewed in the web browser; and

sending the second portion of the rendered content to the web browser.

2. The method of claim **1**, wherein the content is capable of being viewed by an application program adapted to create the content, and wherein the content is incapable of being viewed by the web browser.

3. The method of claim **1**, the method further comprising:

while sending the second portion of the rendered content to the web browser, transforming a third portion of the content into a fourth portion of the rendered content; and

sending the fourth portion of the rendered content to the web browser.

4. The method of claim **1**, wherein the representation comprises a full-fidelity representation of the content.

5. The method of claim **1**, wherein in response to receiving the request to view the content, transforming the first portion of the content into the second portion of rendered content comprises:

determining whether the rendered content is available in a content store;

in response to determining that the rendered content is available in the content store, retrieving the rendered content from the content store and sending the rendered content to the web browser; and

in response determining that the rendered content is not available in the content store, transforming the first portion of the content into the second portion of rendered content.

6. The method of claim **1**, wherein receiving the request to view the content from the web browser comprises receiving the request to view the content from the web browser over a communications network; and

wherein sending the second portion of the rendered content to the web browser comprises sending the second portion of the rendered content to the web browser over the communications network.

7. The method of claim **1**, the method further comprising determining whether a user operating the web browser has permission to view the content; and

wherein in response to receiving the request to view the content, transforming the first portion of the content into the second portion of rendered content comprises in response to receiving the request to view the content and determining that the user has permission to view the content, transforming the first portion of the content into the second portion of the rendered content.

8. The method of claim **1**, wherein receiving the request to view content from the web browser comprises receiving a first request to view first content from a first web browser and receiving a second request to view second content from a second web browser;

wherein in response to receiving the request to view the content, transforming the first portion of the content into the second portion of rendered content comprises in response to receiving the first request to view the first content and the second request to view the second content, transforming a third portion of the first content into a fourth portion of first rendered content and transforming a fifth portion of the second content into a sixth portion of second rendered content; and

wherein sending the second portion of the rendered content to the web browser comprises sending the fourth portion of the first rendered content to the first web browser and sending the sixth portion of the second rendered content to the second web browser.

9. The method of claim **1**, wherein at least a portion of the rendered content comprises Hypertext Markup Language (HTML).

10. The method of claim **1**, wherein the content comprises a document.

11. A computer-readable storage medium comprising computer-executable instructions stored thereon which, when executed by a computer, cause the computer to:

receive a request to view content from a web browser;

when the request to view the content has been received, transform a first portion of the content into a second portion of rendered content, wherein the rendered content is adapted to provide a representation of the content capable of being viewed in the web browser; and

send the second portion of the rendered content to the web browser.

12. The computer-readable storage medium of claim **11**, wherein the computer-readable storage medium comprises further computer-executable instructions stored therein which, when executed by the computer, cause the computer to:

while the second portion of the rendered content is sent to the web browser, transform a third portion of the content into a fourth portion of the rendered content; and

send the fourth portion of the rendered content to the web browser.

13. The computer-readable storage medium of claim **11**, wherein to receive the request to view content from the web browser, the computer-executable instructions, when executed by the computer, cause the computer to receive a first request to view first content from a first web browser and receive a second request to view second content from a second web browser;

wherein to when the request to view the content is received, transform the first portion of the content into the second portion of rendered content, the computer-executable instructions, when executed by the computer, cause the computer to when the first request to view the first content and the second request to view the second content are received, transform a third portion of the first content into a fourth portion of first rendered content and transform a fifth portion of the second content into a sixth portion of second rendered content; and

wherein to send the second portion of the rendered content to the web browser, the computer-executable instructions, when executed by the computer, cause the computer to send the fourth portion of the first rendered content to the first web browser and send the sixth portion of the second rendered content to the second web browser.

14. The computer-readable storage medium of claim **11**, wherein the representation comprises a full-fidelity representation of the content.

15. The computer-readable storage medium of claim **11**, wherein the content is capable of being viewed by an application program adapted to create the content, and wherein the content is incapable of being viewed by the web browser.

16. A rendering computer system comprising:

a processor;

a memory communicatively coupled to the processor; and

a program module which executes in the processor from the memory and which, when executed by the processor, causes the processor to

receive, via a communications network, a request to view content from a web browser,

when the request to view the content has been received, transform a first portion of the content into a second portion of rendered content, wherein the rendered content is adapted to provide a representation of the content capable of being viewed in the web browser, and

send, via the communications network, the second portion of the rendered content to the web browser.

17. The rendering computer system of claim **16**, wherein the program module, when executed by the processor, further causes the processor to:

while the second portion of the rendered content is sent to the web browser, transform a third portion of the content into a fourth portion of the rendered content; and

send, via the communications network, the fourth portion of the rendered content to the web browser.

18. The rendering computer system of claim **16**, wherein to receive the request to view content from the web browser, the program module, when executed by the processor, further causes the processor to receive a first request to view first content from a first web browser and receive a second request to view second content from a second web browser;

wherein to when the request to view the content is received, transform the first portion of the content into the second portion of rendered content, the program module, when executed by the processor, further causes the processor to when the first request to view the first content and the second request to view the second content are received, transform a third portion of the first content into a fourth portion of first rendered content and transform a fifth portion of the second content into a sixth portion of second rendered content; and

wherein to send the second portion of the rendered content to the web browser, the program module, when executed by the processor, further causes the processor to send the fourth portion of the first rendered content to the first web browser and send the sixth portion of the second rendered content to the second web browser.

19. The rendering computer system of claim **16**, wherein the representation comprises a full-fidelity representation of the content.

20. The rendering computer system of claim **16**, wherein the content is capable of being viewed by an application program adapted to create the content, and wherein the content is incapable of being viewed by the web browser.

* * * * *