(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
28 September 2017 (28.09.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/165566 A1

(51) International Patent Classification:
A61B 34/10 (2016.01) *A61B 19/00* (2006.01)
A61B 5/00 (2006.01) *G06K 9/34* (2006.01)

(21) International Application Number:
PCT/US2017/023669

(22) International Filing Date:
22 March 2017 (22.03.2017)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/313,530 25 March 2016 (25.03.2016) US

(71) Applicant: **THE REGENTS OF THE UNIVERSITY OF CALIFORNIA** [US/US]; 1111 Franklin Street, 12th Floor, Oakland, California 94607-5200 (US).

(72) Inventor: **BRUNICARDI, F. Charles**; University of California, Los Angeles, Department of Surgery, 1328 16th Street, 2nd Floor, Santa Monica, California 90404 (US).

(74) Agent: **O'BANION, John**; O'BANION & RITCHEY LLP, 400 Capitol Mall, Suite 1550, Sacramento, California 95814 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK,

[Continued on next page]

(54) Title: HIGH DEFINITION, COLOR IMAGES, ANIMATIONS, AND VIDEOS FOR DIAGNOSTIC AND PERSONAL IMAGING APPLICATIONS

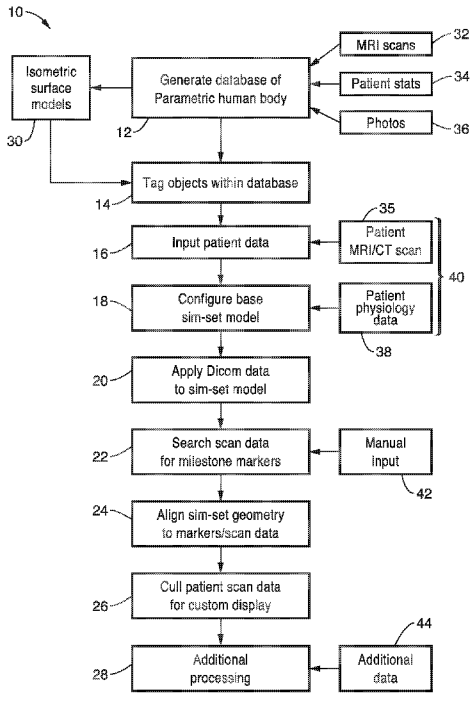


FIG. 1

(57) Abstract: High definition, color images, animations, and videos for diagnostic and personal imaging applications are described along with methods, devices and systems for creating the images, as well as applications for using the images, animations and videos.

WO 2017/165566 A1

SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

HIGH DEFINITION, COLOR IMAGES, ANIMATIONS, AND VIDEOS FOR DIAGNOSTIC AND PERSONAL IMAGING APPLICATIONS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to, and the benefit of, U.S. provisional patent application serial number 62/313,530 filed on March 25, 2016, incorporated herein by reference in its entirety.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not Applicable

INCORPORATION-BY-REFERENCE OF COMPUTER PROGRAM APPENDIX

[0003] Not Applicable

NOTICE OF MATERIAL SUBJECT TO COPYRIGHT PROTECTION

[0004] A portion of the material in this patent document is subject to copyright protection under the copyright laws of the United States and of other countries. The owner of the copyright rights has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the United States Patent and Trademark Office publicly available file or records, but otherwise reserves all copyright rights whatsoever. The copyright owner does not hereby waive any of its rights to have this patent document maintained in secrecy, including without limitation its rights pursuant to 37 C.F.R. § 1.14.

BACKGROUND

[0005] 1. Field of the Technology

[0006] This technology pertains generally to imaging modalities used in healthcare, and more particularly to a transformational imaging and

simulation platform that extends photorealism to healthcare and other uses.

[0007] 2. Background Discussion

[0008] The primary purpose of most medical imaging modalities is to help physicians of all specialties, as well as patients, provide accurate diagnoses to improve therapeutics. The beginning of modern imaging began with William Rontgen and the discovery of x-rays in 1895. The advent of CT scanning, MRI scanning, Ultrasound, PET scanning and other computer generated imaging has provided a remarkable advance in healthcare worldwide over the last thirty years that greatly enhanced the ability of healthcare providers to diagnose and treat diseases of all types, as well as avoid unnecessary and costly procedures, such as exploratory laparotomies.

[0009] While these advances are laudable, the quality of the images for almost all imaging modalities produced is still remarkably primitive. The images being produced in healthcare are currently presented in primitive black and white images and are difficult to interpret except by few highly trained specialists, such radiologists, nuclear medicine specialists and imaging technicians. There are attempts using relatively primitive color schemes in low definition and in 3-D. Additionally, low definition animations of surgeries and other procedures, such as those made by DaVinci Robot or Simbionix, do exist and have been demonstrated to have limited training value for surgical trainees. There are medical video games with low definition, unrealistic animations, such as Atlus' Trauma Center Series, however these games are not practical for medical and surgical training purposes.

[0010] Recent advancements have been made in technologies relating to photo-real render engines, which primarily depend on the use of shader code applied to an isometric surface model. Generally, CT scans are volumetric and are designed for clinical viewing only. While converting volumes to isometric surfaces is possible with existing 3rd party applications, the converted surface lacks detail with respect to information about the physical properties of the materials relating to the visible

wavelength – color, glossiness, etc.

[0011] Raw scan data from current imaging modalities also isn't high resolution due to limitations with the current generation of scanners. Additionally, the raw data can be very noisy with lots of artifacts that will be in direct conflict with generating photo-real rendering. Most scans are just done to the localize area of interest, so getting a full body scan to build isn't practical.

[0012] Current technology would require a 3rd party company to process the scan data and manually make a photo-real rendering of it, which would be cost prohibitive and overly time consuming (e.g. weeks of time to produce).

[0013] Furthermore, some scanners use custom file formats that only work with their software, making it hard to process to the data to make a photo-real rendering.

[0014] Even with the current advances in MRI and CT scanners, the data is presented in a purely clinical manner, which can be very confusing and intimating to even the trained eye, but to the patient it's even more overwhelming.

[0015] Accordingly, creating a new tool that can alleviate these issues and provide doctors, surgeons, radiologists, teachers, and patients a new means to view the data is vitally important.

BRIEF SUMMARY

[0016] One aspect of the present disclosure is a transformational imaging and simulation platform, herein referred to as "SIMaging."

[0017] Another aspect is systems and methods to convert a conventional two dimensional black and white medical image into a colorized "photorealistic" image, and automatically render diagnostic information based on the photorealistic image.

[0018] Actual photorealism, color images of the patient's body, organs, tissues and cells transforms diagnostic capability for the radiologist and healthcare providers of all specialties, as well as education of future

healthcare providers and the patients themselves. In turn, these personalized images may be used for real time simulations coupled with artificial intelligence that may be used for practice of any intervention/operation, as well as realtime use during virtually any medical procedure to guide surgeons and interventionalists of all specialties, including all surgical specialties, gastroenterology, radiology, pulmonary, anesthesia, pain medicine, cardiology, etc, through the procedures.

[0019] The images and simulations produced from the present technology may be used on a daily basis to transform imaging, diagnostic capabilites and therapy in all hospitals, centers, clinics and medical schools worldwide. The system may also be connected to artificial intelligence to guide the surgeon/interventionalist through procedures, like a GPS system, in real time, while documenting each step of the procedure according to standard operating protocols, thus standardizing and documenting procedures and entering the information into databases in realtime. The images and simulations can be used on a daily basis to transform the education of surgeons and interventionalists of all specialities and their trainees in all medical universities and hospitals and clinics worldwide. These advances improve quality of care, improve documentation and lower costs of complications, readmissions and documentation.

[0020] The personalized images produced from the technology can also be used for home health systems that would help patients monitor such data as their weight, body mass, body fat content, body water content, body muscle content, which can be connected to an electronic medical record to assist in health maintenance, as well as to track response to therapy and recovery from operations over time. Users visualize and save images of their body images over time. Expecting women (and their partners) may visualize photorealistic images of their babies *in utero*. The personalized images, coupled with artificial intelligence, help guide personal grooming, make-up applications and dress for all different types of events. The personalized images and simulations may also be used for entertainment purposes for the movie, television, internet, and gaming industries. The

platform may further be used to evaluate employees before and during employment, for example, athletes, on a global basis.

[0021] In one implementation, the technology of the present description is incorporated into imaging machines for hospitals, clinics, operating rooms, interventional suites and medical schools and laboratories, worldwide. Further embodiments include systems comprising home health units using computer gaming systems such as Kinnect, Wii, Oculus Rift, and Playstation to assist in home health as well as personal grooming and gaming using computer and robotic avatars. The technology of the present description may also be used for movie, television, music videos and internet. In doing so, this would represent the next generation in personalized imaging and therapy to enhance healthcare, education and entertainment on a global scale.

[0022] Another aspect of the technology described herein is to combine the photorealism imaging technology with enhanced molecular contract technology. The ability to produce photorealism imaging, in high definition, with enhanced molecular contrast, and the production of personalized animated simulations from these images represents the benchmark transformation of personalized medicine and surgery on a global scale. Therefore, the use of SIMaging represents a transformation in health maintenance and healthcare delivery that will revolutionize the quality of life and longevity of mankind, as we enter into the age of personalized medicine and surgery.

[0023] Another aspect of the technology described herein is the use of tissue specific gene delivery-imaging platform to enhance the imaging differences between healthy tissues and cells, and diseased tissues and cells, using state-of-the-art “molecular contrast agents.” In doing so, personalized photorealism renderings and simulations of the healthy and diseased body, organs, tissues and cells can be produced to assist both health care providers of all specialties and humans worldwide to evaluate their overall body health, diagnose their diseases of all types, practice, plan and assist in operations and therapeutic interventions in unlimited

applications, as well as help track recovery from therapies, operations and interventions.

[0024] The photorealism images produced by the technology can be used for real time personalized animated simulations that will assist surgeons and other interventionalists of all specialties to improve their quality of care for their patients. These personalized animated simulations of both healthy and diseased organs, tissues and cells will assist health care providers of all specialties worldwide to evaluate their patients overall body health and diagnose their diseases of all types. The personalized animated simulations will also enable surgeons and other medical practitioners of all specialties to practice and plan their proposed procedures on their patients' images before the operation or procedure. Furthermore, the personalized animated simulations coupled with artificial intelligence and robotic assistance can be used during the procedure using specialized glasses or computer screens to assist surgeons and interventionalists with the procedures in real time to improve quality of care, reduce complications and reduce costs of healthcare by guiding the surgeon and interventionalist using standard-of-care protocols while capturing the details of the operation or intervention in real time and into electronic healthcare databases. The personalized animated simulations and artificial intelligence with robotic assistance will assist in standardization and documentation of all procedures in real time. In doing so, the quality of care and quality of life for the patients, as well as quality of life for the healthcare providers, may greatly improved, while significantly lowering healthcare costs.

[0025] The personalized animated simulations can also be used for educational purposes for health care providers, future healthcare providers and patients. The personalized animated simulations may used in home health systems to help patients evaluate their own health status, as well as disease states and recovery from therapies, operations and other interventions.

[0026] The SIMaging technology can also be extended to entertainment purposes, such as movies, television, music videos and internet computer

gaming of all types, as well as to personal grooming applications and simulated aging by using computer and robotic avatars. The technology can be used by private industry to evaluate employee health status before and during employment, such as the sports industry worldwide. The technology can also be used for farming husbandry purposes. Furthermore, the technology can be used for research purposes in clinics and laboratories worldwide. Simulated procedures can be carried out and meaningful predictions can be made prior to use of lab animals or appropriate human participants.

[0027] Further aspects of the technology will be brought out in the following portions of the specification, wherein the detailed description is for the purpose of fully disclosing preferred embodiments of the technology without placing limitations thereon.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWING(S)

[0028] The technology described herein will be more fully understood by reference to the following drawings which are for illustrative purposes only:

[0029] FIG. 1 is a schematic flow diagram of a method for generating a parametric simulated model (SIM-SET) of an anatomical region of a patient in accordance with the present description.

[0030] FIG. 2 is a system for outputting the SIM-SET of FIG. 1 based on input patient data.

[0031] FIG. 3 illustrates enhancing an MRI of the brain according to an embodiment of the technology described herein.

[0032] FIG. 4 illustrates enhancing an angiogram of an aortic aneuryism with an endovascular stent according to an embodiment of the technology described herein.

[0033] FIG. 5 shows a schematic flow diagram of a model for personalized genomic medicine and surgery according to embodiments of the technology described herein.

[0034] FIG. 6 illustrates a flow diagram of a process flow for enhanced

imaging using promoters according to an embodiment of the technology described herein.

[0035] FIG. 7 illustrates an embodiment of molecular contrast imaging according to the technology described herein.

[0036] FIG. 8A and FIG. 8B illustrates an example of enhanced imaging of pancreas cancer using promoters according to an embodiment of the technology described herein.

[0037] FIG. 9A through FIG. 9D illustrate an example of enhanced imaging of a tumor in mice using promoters according to embodiment of the technology described herein.

[0038] FIG. 10A and FIG. 10B show a raw scan and isosurface render, respectively, using CT data.

[0039] FIG. 10C and FIG. 10D show a raw scan and isosurface render, respectively, for US data.

[0040] FIG. 11A through FIG. 11C show a flow diagram for a processing method for automatic high-quality rendering of arbitrary human dicom scans with a virtual camera.

DETAILED DESCRIPTION

[0041] **A. SIMaging System and Method Overview**

[0042] The technology described herein is directed to transformational imaging and simulation platforms, "SIMaging," applied to current medical imaging modalities to greatly enhance the accuracy and interpretability of medical images. The technology facilitates accurate diagnoses, improved therapeutics, and enhanced training for healthcare professionals, patients and others.

[0043] SIMaging provides novel methods to present medical CT/MRI scans (and other acquired data sets such as X-ray, photos, etc.) to patients and doctors, allowing them to see in a very easy to decipher photorealistic real-time rendering presentation by creating new geometry.

[0044] SIMaging is based largely on development of a "SIM-SET," which is a parametrically built representation of the human body, or portion of a

human body, with fully photo-real geometry and shaders that can be rendered in real-time. The SIM-SET is automatically aligned and calibrated to one or more sources of input data such as: MRI/CAT scans, weight, BMI, height, x-rays, photos, manual input, etc. This dynamically creates a 3D patient SIM-SET to an almost exact representation of the real patient with all of the pre-defined photo-real characteristics built into it for generating imagery with realistic looking skin, bone, and flesh using a data base of BRDF (bidirectional reflectance distribution function) shaders with sub-surface scattering.

[0045] FIG. 1 shows a schematic flow diagram of a method 10 for generating a parametric simulated model (SIM-SET) of an anatomical region of a patient in accordance with the present description. First, a database of a parametric human body (SIM-SET), or anatomical portion of the body, is created at step 12 by acquiring input data in the form of one or more of imaging scans 32 (e.g. MRI/CT/US scans), patient statistics 34, photos 36, etc. The SIM-SET comprises mainly of volume data and/or isometric surface models 30 of one or more aspects of the anatomy (e.g., for both male and female patients), the surface models 30 being designed for photo-real real-time VR rendering.

[0046] Next at step 14, each object in the SIM-SET is 'tagged.' The tagged data is configured so that any piece of the human parametric model can be turned on or off for viewing (e.g. to show skeletal, or vascular and muscle together, etc).

[0047] The data from steps 12 and 14 may be stored in a secure dedicated database, piggy back onto existing hospital systems, or may be completely integrated into a manufacture's database (such as the scanner manufacturer, GE, or the like).

[0048] With the database established, the system (see, e.g., system 100 shown in FIG. 2) is ready to be applied to individual patients. At step 16, individual patient data is input, which may be in the form of an MRI/CT scan 35, or the like.

[0049] At step 18, a patient specific, base SIM-SET model is developed

from the database as a function of input patient physiology data 38, e.g. specific physical characteristics of the patient. Both the patient scan 35 and physiology data 38 are patient data 40 that is specific to the individual patient.

[0050] At step 20, DICOM data/files (preferably non-proprietary) are extracted from a CT/MRI scanners.

[0051] At step 22 the scan data is searched for specific milestone markers of anatomy that will later be used to align the SIM-SET geometry. Manual input 42 may be used for selecting one or more milestone markers if a patient has a unique anatomy, e.g. birth defects, amputations, or other abnormalities.

[0052] At step 24 the markers are then used to perform isometric surface alignment of the SIM-SET geometry to the patient's scan 35. In a preferred embodiment, the alignment is done taking into account other input data 38 such as weight, height, BMI, X-ray, photos, etc. This alignment is performed on both volume data and surfaces, and can also be adjusted by manual input 42 via selecting the key structures on different slices of a MRI/CT scan to help fine-tune the SIM-SET.

[0053] Using the tags from the SIM-SET, the patient scan data 35 can now be 'culled' at step 26 for rendering and custom display via a fully controlled viewing system for showing all the data in a photo-real presentation.

[0054] Additional processing may be performed at step 28 as a function of further input 44. For example, photographic reference of skin color can be input to make the exact tone and hue of the patient. Now that that a fully 3D model of the patient's body exists, other data can now be connected to it, e.g. auto-alignment and projection of wound, surgeon markups, or even X-rays (for example) directly onto the surface of the body, essentially generating a 3D photo gallery based on body location. Notes and other text files can also be connected to different locations on the body, giving the patient and the doctor a fully 3D representation of data.

[0055] Referring to the system 100 shown in FIG. 2 the output SIM-SET or simulation model may be rendered in real-time for view on a display 112.

System 100 further comprises a computer/server 102 comprising a processor 106 and memory 108 for storing application programming 102. Wherein programming 102 comprises instructions configured for execution by the processor for transforming the input patient data 40 according to method 10 into the output simulation model 110 for display or with VR assisted headgear (not shown).

[0056] Programming 110 is furthermore adaptable so that doctors and patients can add data to the SIM-SET over time. For example, additional scans 35 can be added at a later date to show progress over time, or a patient could take photos of a post-op condition and apply them to the model data set. Tracking a patient's weight loss may also be represented in an animated time-lapse photo-real presentation.

[0057] Because the data SIM-SET has anatomy information that is not present in the patient scans (it can extrapolate the entire body based off of just a chest CT scan and some other input data, for example), it may also serve as an educational reference for the entire body, and can be viewed and explored as such. Hence the name “Photo-real Simulation-Visualography” because of the extrapolation of un-scanned portions of the body.

[0058] Because the clinical scan data 35 is still maintained in the SIM-SET, it can be viewed at the same time if needed. This allows the physician to show raw data to a patient in combination with photo-real rendering.

[0059] Furthermore, because lighting and reflections are integral to photo-real rendering, any photo-real HRDI (high range dynamic image) environments can be selected for viewing the SIM-SET within, such as: a generic modern doctors office, abstract white cyc, park setting, etc. Several environment presets can be included and used depending on the desired presentation of the SIM-SET.

[0060] The SIMaging systems and methods of the present description may be integrated into the medical field via several different avenues, such as working directly with manufactures to include it with their MRI/CT scanners, or to have it be a stand-alone 3rd party application that patients or doctors

can use from home or at school.

[0061] The SIMaging systems and methods of the present description capable of producing color high definition photographs (photorealism) and animated simulations of any person's body, organs, tissues and cells from the images obtained from standard imaging modalities, such as CT scans, MRI, ultrasound, PET scans, brain scans, nuclear medicine scans, optical imaging scans, and other targeted imaging technology, using high definition animation platforms.

[0062] SIMaging systems and methods of the present description may be expanded to be the foundation for visualizing not just scan data (MRI, CT, etc.), but any other input data that benefits from alignment with a 3D photo-real representation of the patient, such as: pre-surgery markup photos, post-surgery follow up photos, X-rays, artificial joints, pace makers, etc.

[0063] This photo real virtual patient may be used to show the results of a recent scan or surgery, or may be used with VR assisted viewing for a surgeon performing an evasive procedure, or finally to a teacher showing students a photo-real anatomy lesson. And as additional data sets are collected over the years, it can also provide a fully visual historical presentation of patient to better track their health and progress.

[0064] The technology of this disclosure can be described in terms of several main embodiments, which will now be described. In one embodiment, referred to herein as Embodiment 1, conventional images from standard diagnostic imaging platforms (e.g., DICOM format) are transformed into high definition colorized, "photorealistic" images using the SIMaging system and method of the present description.

[0065] In another embodment, referred to herein as Embodiment 2, three dimensional animations are created from the photorealistic images.

[0066] In another embodiment, referred to herein as Embodiment 3, the photorealistic images are rendered in such a way as to depict medical conditions of interest. For example, an image of tissue or an organ would visually depict cancerous areas in the tissue or organ.

[0067] It will be appreciated that the technology described herein includes

other embodiments as described herein and as would be appreciated to those skilled in the art in view of the description herein.

[0068] **B. General Embodiment 1**

[0069] In General Embodiment 1, conventional images from standard diagnostic imaging platforms (e.g., DICOM format) are transformed into high definition colorized, "photorealistic" images using the systems and methods detailed above.

[0070] For example, animated platforms of both cartoon as well as photorealism movies may be generated using computer generated imaging facial recognition technology to enhance the standard diagnostic images that are produced by the different imaging modalities. Preferably this system would implemented in close collaboration between the surgeons, interventionalists of all procedures, radiologists, animators and software developers to combine the use of medical knowledge of the human body, both anatomy and physiology, and therapeutic interventions with computer animation platforms to produce photorealism images and simulations that are created for the benefit of healthcare.

[0071] For example, a CT/MRI/US scan of a patient's breast and/or internal gastrointestinal organs may be input as scan 35 (see FIG.1) or patient data 40 (FIG. 2) into system 100 and is transformed into output simulation model 110, e.g. as high definition colorized, "photorealistic" images or animations in collaboration with, for example, surgeons and other interventionalists who can interpret the images and their relevance to health and disease. This may be performed for both normal and healthy tissues versus those tissues that are diseased. Normal and healthy breast or appendix images may be used for the initial images to create a photograph of the breast or appendix similar to that seen during an operation. Once the normal breast or appendix is visualized, different phases of the diseased breast or inflamed appendix can be animated using input CT scans, MRI scans, and ultrasounds of patient's breast or inflamed appendices, which are correlated with the images seen in the operating room during open operations, as well as laparoscopic appendectomies. The same can be

applied for gallbladders that are both normal as well as inflamed. For both breast or appendix, the still or animated images could progress to cancer of the breast or perforated appendix with surrounding purulent material. Once the fundamentals are developed for these organs, the technology can then be applied to any disease process that can be imaged such as cardiovascular diseases, cancer, arthritis, chronic inflammatory diseases, and any range of normal tissues versus pathologic states that can be imaged.

[0072] In another example, CT/MRI/US scans of the body are performed on a patient. The information for each organ seen on the primitive images of the CT/MRI/US is evaluated and compared with known images and photographs of the human body and organs from anatomy atlases and other documented sources. Using, for example, the expertise of a surgeon or other interventionalist, or information in a database of related information previously developed, the images of the organs on the CT/MRI/US scan of the patient's body are transformed into a high definition, photorealistic rendering of the internal gastrointestinal organs using the system and methods of the present description. Initially, the program can be focused on one organ, such as the liver or appendix, to simplify the process. Cross-references of the animation being produced with the CT scan may done on a continuous basis to ensure accuracy of the photo being produced. Once completed, the CT/MRI/US scan may be reviewed and compared with the photorealism rendering of the rendered animation image. A determination may be made of whether the system is able to reproduce the photorealism images from the DICOM format CT/MRI/US scan without the assistance of the animators and surgeon. Once this is accomplished, a database of all internal organs is developed and tested against CT/MRI/US scans from multiple patients. This process may be repeated for other imaging modalities of known patients, such as PET-CT and nuclear medicine scans.

[0073] Simultaneously, the same patient's head and body may scanned into the animation computer system using a 4 camera scanning facial recognition technology such as that provided by Hydraulx, Inc. The

CT/MRI/US scan may be reviewed in collaboration with the surgeon, radiologist and the animation team and the information then transferred into the system (e.g. using NECTAR and Dell computers animation software, such as Poser Pro 2010 3D Modeling & Animation). The information of each organ seen on the primitive images of the CT/MRI/US may be discussed in detail and compared with known images and photographs of the human body and organs from anatomy atlases and detailed information obtained from other documented sources. With the guidance of the surgeon, the images of the organs on the CT/MRI/US scan of the patient's body are then transformed into a high definition photorealism rendering of the internal gastrointestinal organs using the system. As before, there may be a focus on one organ, such as the liver or appendix, to simplify the process. Cross-references of the animation being produced with the CT/MRI/US scan may done on a continuous basis to ensure accuracy of the photo being produced. Once completed, the image of the CT/MRI/US scan will be reviewed and compared with the photorealism rendering. A determination may be made of whether the system is able to reproduce the photorealism images from the DICOM format CT/MRI/US scan without the assistance of the animators and surgeon. Once this is accomplished, the database of all internal organs may be developed and tested against other patients' CT/MRI/US scans. This process will be repeated for other imaging modalities of known patients, such as PET-CT and nuclear medicine scans.

[0074] It will be appreciated, therefore, that an aspect of the technology described herein is to create high definition colorized animated images that are accomplished by collaboration of the system of the present description with diagnostic imaging platforms from CT scans, MRI, ultrasound, mammography, brain scans, nuclear medicine scans, PET scans to enhance diagnostic capability and therapy for healthcare providers and education of patients.

[0075] In one exemplary configuration, FIG. 3 illutstrates enhancing an MRI of the brain according to an embodiment of the present technology. An MRI

scan 40a is input into the system 100 to generate and output one or more photorealistic simulations 26a of the brain.

[0076] FIG. 4 illustrates another exemplary embodiment for enhancing an angiogram of an aortic aneuryism with an endovascular stent according to an embodiment of the technology. In such embodiment, angiogram 40b is input into the system 100 to generated and output photorealistic simulation 26b of the stent and surrounding anatomy in the chest cavity.

[0077] Another exemplary embodiment may entail inputting a standard image of fetal ultrasound to generate a photorealism rendering of the baby within the mother's womb *in utero*. A further exemplary embodiment may include inputting a standard image of breast seen on mammogram and/or MRI scan to generate an animated image of the breast, or inputting a standard image of liver cancer seen on CT scan to generate photorealism images of liver cancer metastases.

[0078] It will also be appreciated that potential levels of images and generated simulation/animation are as follows: 1) whole body, face and skull, 2) organs, 3) cells, 4) molecular pathways and functional genomics, 5) atomic and 6) subatomic.

[0079] **C. General Embodiment 2**

[0080] In this embodiment, animated simulations are generated from the personalized images. The result can be used for medical and surgical simulation video games for training of interventionalists of all specialities. Animated simulation for any interventional procedures would help practitioners of all specialties practice prior to the actual procedure and also help guide the practitioner in real time through the procedure using artificial intelligence to track milestones of the procedure, as well as track the progress of the milestones for the procedure in real time in the medical records. This would lead to standardization of procedures on a global scale and improve outcomes and quality of care. In addition, practitioners would also avoid the need for costly and often inaccurate dictations thus improving documentation of healthcare.

[0081] System 100 configured in the form of a home animation imaging

system would provide information that would help patients recover from any procedure and also help guide the patient, in real time, through the recovery process using artificial intelligence to track milestones of the recovery, as well as tracking the progress of the milestones for the recovery in real time in their medical records. Data that may be recorded and analyzed by the patient includes, but is not limited to:

(a) weight loss;

(b) weight gain;

(c) general health at home;

(d) fat content;

(e) muscle mass;

(f) water content.

[0082] Such a system may enable standardization of recovery from procedures on a global scale and improve outcomes, quality of care and quality of life. This would also serve avoid the need for costly readmissions and improve documentation of home healthcare.

[0083] The home animation imaging system may be used to facilitate personal grooming, including application of facial make-up, other cosmetic applications, dress, etc., in 3D for a number of special events.

[0084] Application programming 104 may be configured as a mobile app for use on smartphone or other mobile devices in the form of gaming systems that involve the human body, repair of the human body, and portraying any injury of the human body such as gunshots, stabbings, car crashes, and other trauma that are currently used in gaming systems. Realistic simulations of trauma to the body may be provides, in addition to the ability to repair the trauma. The games could be used to help gamers of all ages that are interested in any medical applications.

[0085] Accordingly, Embodiment 2 includes animated simulations from the high definition colorized, photorealism images described in Embodiment 1 above, using the system 100 as applied to standard diagnostic imaging platforms (e.g. in DICOM format).

[0086] Following the generation of high definition colorized, photorealism

images from the patient's CT/MRI/US scan of the body, or from any standard imaging modality, the original CT/MRI/US scan and the photorealism images may be reviewed. The information for each organ seen on the primitive images of the CT/MRI/US is compared with the images of the patient's body and organs along with personal knowledge of the surgeon and interventionalist of the specific details of the procedure or intervention to be performed. Identification of the actual human anatomy and color from selected black, white and grey features of the CT/MRI/US images, blood vessels, fat, liver parenchyma is preferably performed done by the surgeon and radiologist using the Hounsfield unit scale, standard contrast agents and molecular contrast agents to differentiate densities of blood, water and parenchyma, tumors, scars, etc., are translated into the simulated images by animators in coordination with the photorealism images of the tissues of the selected organs and tissues.

[0087] With the guidance of the surgeon and interventionalist, the simulation of the procedure may be developed from actual high definition colorized images of the organs intended for the operation or procedure from the CT/MRI/US scan of the patient's body using the animation platform. Initially, there can be a focus on developing a simulation of one organ, such as the appendix, to simplify the process. Cross-references of the animated simulation being produced with the CT/MRI/US scan may be done on a continuous basis to ensure accuracy of the simulation being produced. Once completed, the animated simulation of the CT/MRI/US scan, or similar modality, will be reviewed and compared with the photorealism rendering by the surgeon, radiologist, interventionalist, etc.

[0088] A determination may be made of whether the system is able to reproduce the photorealism images from the DICOM format CT/MRI/US scan without the assistance of the animators and surgeon. The animated simulations may ultimately be reviewed by the entire team, which will ensure accuracy of the automated simulations. Once this is accomplished, a database of animated simulations of all operations and interventions is developed and tested against other patients' CT/MRI/US scans. The

animated simulation may be evaluated using practice operation or interventions by the surgeon or interventionalist, respectively, to determine the usefulness of the simulation in helping prepare for the actual operation or intervention.

[0089] Once evaluated, the animated simulation may be evaluated again in real-time during the actual operation or intervention using picture-in-a-picture technology to determine the usefulness of the simulation in assisting the surgeon or interventionalist during the actual operation or intervention. The animated simulations may be continuously modified to incorporate artificial intelligence technology and real-time data capture technology to assist the surgeon and interventionalist with the performance and reporting of the details of the operation or intervention. The goal is to ensure safety and improve quality of care by using standard of care protocols that are captured using artificial intelligence as the operation or intervention proceeds in real time. This process may be repeated for other imaging modalities of known patients, such as PET-CT or nuclear medicine scans.

[0090] In another example, the same patient's head and body is scanned simultaneously into the system (e.g., using a 4 camera scanning facial recognition technology, as provided by Hydraulx, Inc.). The CT/MRI/US scan is then reviewed in collaboration with the surgeon, radiologist and the animation team and the information transferred into the system (e.g. using NECTAR and Dell computers animation software, such as Poser Pro 2010 3D Modeling & Animation). The information of each organ seen on the primitive images of the CT/MRI/US may be discussed in detail and compared with known images and photographs of the human body and organs from anatomy atlases and detailed information obtained from other documented sources. With the guidance of the surgeon, the images of the organs on the CT/MRI/US scan of the patient's body are then transformed into a high definition photorealism rendering of the internal gastrointestinal organs using the system.

[0091] The original CT/MRI/US scan and the photorealism images may be reviewed in collaboration with the surgeon, radiologist, interventionalist and

the animation team and the information will be transferred into the animation simulation platform for development of the animated simulation. The information of each organ seen on the primitive images of the CT/MRI/US is discussed in detail and compared with the images of the patient's body and organs along with personal knowledge of the surgeon and interventionalist of the specific details of the procedure or intervention to be performed. Identification of the actual human anatomy and color from selected black, white and grey features of the CT/MRI/US images, blood vessels, fat, liver parenchyma is performed by the surgeon and radiologist using the Hounsfield unit scale, standard contrast agents and molecular contrast agents to differentiate densities of blood, water and parenchyma, tumors, scars, etc., and will be translated into simulated images by the animators in coordination with the photorealism images of the tissues of the selected organs and tissues.

[0092] The information and databases, as well as the technology from these processes can also be implemented for home health systems that are connected to electronic medical records, for gaming purposes, for movies and television and for personal grooming programs.

[0093] **D. General Embodiment 3**

[0094] As an extension of Embodiment 1, the systems and methods of the present descriptin mayb be used for functional genomics and molecular imaging to develop "molecular contrasts" for SIMaging.

[0095] PET-CT scan imaging currenlty exits, but the images are of low definition. The quality of definition can be greatly enhanced using the photorealism imaging system 100 of the present description, as well as the ability to image any diseased tissue versus healthy, normal tissue. Similar to that of intravenous and oral contrast agents used in current imaging modalities, the present system would use "molecular contrast" to differentiate different healthy cells and tissues, as well as diseased cells and tissues, from one another using tissue specific- synthetic promoter-driven gene delivery platform to deliver cell and tissue specific imaging genes.

[0096] FIG. 5 shows a schematic flow diagram of a model 150 for personalized genomic medicine and surgery according to embodiments of the technology described herein. First genomic evaluation is performed at block 152 in the form of one or more of exome or gene sequencing, proteomics, etc. SIMaging method 10 is then performed at block 154 via one or more imaging modalities (e.g. MRI, CT, US, PET, targeted molecular imaging, nuclear medicine, optical imaging, etc.). Functional studies of targets and/or mutations, such as overexpressed proteins, oncogenes, tumor suppressor genes, signaling targets, etc., is then performed at block 156. Finally, targeted therapy (e.g. preventative, standard, and targeted therapies, preclinical studies, clinical trials, etc.) are preformed at block 158.

[0097] FIG. 6 illustrates a flow diagram of a process flow 200 for enhanced imaging using promoters (i.e. molecular contrasts for SIMaging) according to an embodiment of the technology described herein.

[0098] The functional genomic analysis system 202 may be performed on all given healthy tissues of the body, as well as diseased tissues for identification of overexpressed Protein X in each given tissue at block 204. This means that that tissue X has the transcriptional mechanisms to cause activation of the promoter of the gene of protein X, thus causing overexpression. For example, the insulin protein is over expressed in healthy islets of Langerans in the pancreas, as well as in insulinoma tumors. Along with others, we have identified that the transcription factor, PDX1, is responsible for activation of the insulin promoter in these healthy and diseased tissues.

[0099] At block 206, the synthetic promoter of Protein X is generated, which will drive gene expression in the healthy and diseased tissues. For example, we generated a synthetic insulin promoter (BL promoter) and have shown that the BL promoter is very efficient at tissue specific delivery of imaging genes in islets, insulinoma tumors cells and cancer cells.

[00100] At block 208, SIMaging is performed on the synthetic promoter. The promoter of protein X is used for delivery of theranostic genes, which can be used for enhanced cell and tissue specific imaging of both healthy

tissues and diseased tissues, as well as therapy. For example, imaging healthy islets of Langerhans within the pancreas (see image 40c in FIG. 7), as well insulinoma tumors (and other cancers) in mice using the BL promoter may be used to drive thymidine kinase and somatostatin receptor subtype 5 genes. These images can then be used to generated photorealism images (see image 26c in FIG. 7), and ultimately simulations of healthy and diseased tissues for personalized SIMaging step 210 based upon the patients imaging studies. As defined in the GIFT model shown in FIG. 5, this system represents personalized medicine, and thus is termed SIMagingi n accordance with the present description.

[00101] The following details the basic science behind the embodiment 200 of FIG. 6. It is to be noted that the BL promoter is labeled as SHIP or "Synthetic Human Insulin Promoter" in this section.

[00102] 1. Using a functional genomics system we have determined that PDX1 is overexpressed in most cancers and can be used as a target to activate the insulin promoter to drive theranostic genes in cancers. PDX1 promotes PDAC via a PDX1 amplifying loop and is a target for insulin promoter driven imaging and therapies.

[00103] 2. We have demonstrated that delivered Rat Insulin Promoter (RIP)-Thymidine Kinase and an analogue of FHBG successfully imaged human pancreatic cancer tumors in mice *in vivo* using optical imaging. For translational purposes, we developed and tested a novel synthetic human insulin promoter (SHIP or BL promoter) utilizing PDX1-activation sites of the human insulin promoter (HIP). Preliminary data demonstrates that SHIP (BL) successfully drives CAT reporter gene expression with significantly higher efficiency than RIP, HIP or CMV promoters in PDX1-positive human pancreatic cancer cells (PANC1), but not in PDX1-negative HPDE cells. We delivered iv SHIP-TK nanoparticles which were successfully expressed in human pancreatic cancer tumors in mice and imaged using optical scanning imaging. The study was repeated on insulinoma tumors in mice and the tumors were successfully imaged. This preliminary data demonstrates that systemically delivered SHIP (BL Promoter) drives gene

expression in PDX1-positive human pancreatic cancer and insulinoma tumors in mice with great efficiency, demonstrating the feasibility of the imaging studies using "molecular contrasts".

[00104] 3. We have shown that the novel SHIP (BL promoter) drives chloramphenicol acetyltransferase (CAT) gene expression with significantly higher efficiency than RIP and HIP in human pancreatic cancer cells. To determine whether SHIP-driven gene expression can be determined using bioluminescence imaging and microPET imaging, SHIP-luciferase is first transfected into pancreatic cancer cell lines with varying PDX1 expression levels, including cell lines PANC1 (PDX1- high), MiaPaCa2 (PDX1-medium) and A549 (PDX1-low), as well as benign HEK293 cells with various doses of PDX1 transfections. Bioluminescence imaging is performed at 24, 48 and 72 hours after gene transfection. The same strategy with imaging TK gene expression driven by SHIP is repeated with microPET imaging using 18F-FHBG. After validation of the accuracy of bioluminescence imaging and microPET imaging in cell lines, these *in vitro* SHIP-luciferase and SHIP-TK imaging experiments are repeated with the presence of bi- shRNAPDX1 or empty-vector NPs, and followed by bioluminescence imaging or microPET imaging at various time points after bi-shRNAPDX1 treatment, respectively. The responses of PDX1-expressing PDAC cell lines to the bi-shRNAPDX1 NP treatments are analyzed and compared with the control groups.

[00105] 4. We have shown that the SHIP promoter drives CAT gene expression with significantly higher efficiency than RIP and HIP in PANC1 cells. To determine whether SHIP-driven gene expression can be imaged using bioluminescence or microPET imaging, SHIP-luciferase or SHIP-TK is first transfected into PDAC cell lines with varying PDX1 expression levels and bioluminescence or microPET (with 18F-FHBG) imaging is performed at 24, 48 and 72 hours after gene transfection. Having experience with RIP-TK imaging in PANC1 subq tumors in SCID mice, test the SHIP-driven gene imaging of PDACis tested *in vivo*. The xenograft tumor models are created as follows:

[00106] i) stably transfected SHIP-TK-PANC1 cells are placed subq in nude

mice; stably transfected SHIP-TK-MiaPaCa2 cells are placed subq in nude mice (n=5 each); the tumors are imaged by microCT after day 30, 60, 90 following implantation of the cells. The subq tumors are measured and the size will be correlation to imaging size. Once the parameters are determined, stably transfected SHIP- TK-PANC1 cells are placed orthotopically in nude mice; stably transfected SHIP-TK-MiaPaCa2 cells will be placed orthotopically in nude mice (n-15 each); the tumors are imaged by microCT after day 30, 60, 90 following implantation of the cells. The orthotopic tumors are measured by sacrificing 5 mice at each time point and the size will be correlated to imaging size. Once these parameters are known, then PANC1 or MiaPaCa2 cells are implanted orthotopically in nude mice (n=15 per group) to test systemically delivered SHIP-TK imaging system. When tumors are measured more than 0.5cm in diameter by microCT, (~30 days after implantation) the mice are given one dose of 35ug of iv injection of SHIP-TK NP, followed by 18F- FHBG then studied using microPET at 24, 48 and 72h after injection to determine whether the PDAC tumor can be detected. The background noise of islets is also be determined.

[00107] ii) Once these parameters are known, along with the data from aim 2a on PDAC tumor volume in KPC mice, KPC mice (n=10) at the optimal age (e.g., 8-10 weeks) receive 35ug of SHIP-TK NP via tail vein and are imaged by microPET following 18F-FHBG at 24, 48 and 72 hours after injection. The mice are then treated with three biweekly cycles of 35ug of mouse-bi- shRNAPDX1 NP. Imaging is repeated 2 weeks after each treatment. Tumors are then harvested to compare tumor volumes, as well as TK and PDX131/46 expression levels.

[00108] iii) These studies are repeated using SHIP-Luc-PANC1 and SHIP-Luc-MiaPaCa2 nude mice model and imaged using Bioluminescence imaging system to image tumors following same protocols as described above. A comparison of these imaging studies may determine the most sensitive and accurate approach for further studies.

[00109] 5. Test data demonstrates that systemically delivered rat insulin

promoter-lacZ (RIP-lacZ), but not CMV- lacZ, resulted in tumor specific expression of LacZ in PDX1-positive metastatic PANC1 tumors harvested from the peritoneal cavity of SCID mice. Furthermore, systemically delivered RIP-Thymidine Kinase and an analogue of FHBG successfully imaged PANC1 subcutaneous tumors in SCID mouse *in vivo* using optical imaging. For translational purposes, we developed and tested a novel synthetic human insulin promoter (SHIP) utilizing PDX1-activation sites of the human insulin promoter (HIP). Preliminary data demonstrate that SHIP successfully drives CAT reporter gene expression with significantly higher efficiency than RIP, HIP or CMV promoters in PDX1-positive PANC1 cells, but not in PDX1-negative HPDE cells. We next developed a SHIP driven Luciferase-RFP (SHIP-Luc2RFP) fusion reporter gene assay and generated stably transfected MiaPaCa2-SHIP-Luc2RFP and PANC1-SHIP-Luc2RFP PDAC cell lines, which can be reliably used to visualize PDX1 gene expression and activity in response to bi- shRNAPDX1 therapy *in vitro*. These cells are used to study PDX1 expression and the mechanisms in aim 1. Having considerable experience with RIP-TK and pro-drug ganciclovir (GCV) therapy in PDAC mouse models, we delivered iv SHIP-TK NPs in a PANC1 xenograft SCID mouse model, which were successfully expressed in PANC1 tumors. When treated with GCV, PDX1-positive PANC1 tumor volume was significantly suppressed with greater efficacy than CMV-TK/GCV or RIP-TK/GCV. These preliminary data demonstrate that systemically delivered SHIP drives gene expression in PDX1- positive PDAC cells and tumors in mice with great efficiency, demonstrating the feasibility of personalized imaging and therapy.

[00110] FIG. 8A and FIG. 8B show an *In vivo* PANC1 tumor-specific imaging using iv RIP-TK/FHBG (FIG. 8A) and RIP-lacZ (FIG. 8B). The image in FIG. 8B is a high definition photorealism of a human pancreatic cancer that was growing in a mouse and was stained blue with a targeted insulin promoter gene delivery system, thus exemplary of SIMaging in accordance with the present description.

[00111] FIG. 9A through FIG. 9D show reporter assay of SHIP (BL) versus

RIP and HIP (FIG. 9A). SHIP-Luc2RFP PDAC cell lines before and after PDX1 knocking down and shown in (FIG. 9B and FIG. 9C. FIG. 9D shows SHIP-TK/GCV successfully targeted and suppressed PDX1-positive PANC1 tumor volume in mice.

[00112] Accordingly, the technology described herein provides an imaging platform that is capable of producing high definition color visual renderings, essentially "photorealistic-type" images, from the two-dimensional black and white images currently obtained from standard tests, such as CT scans, MRI, ultrasound, brain scans, nuclear medicine scans, PET scans, and other targeted imaging technology. These images can be used alone, or can in turn be used for creating real-time simulations that will assist healthcare professionals of all specialties improve their quality of care for their patients,. These real-time simulations, or animations, can further be used for the standardization and/or documentation of that care in real time

[00113] **E. Example Data Extraction**

[00114] A simulation was performed to build a basic pipeline to import raw image scan data from current generation medical imaging devices along with a visualization toolset.

[00115] Raw data was acquired from an ACUSON Sequoia 512 ultrasound system at somewhat lower resolution (256 x 256 x 32 voxels compared to 221 x 251 x 143 voxels on a GE E6 system or about 25% of the measurement points. A higher resolution CT scan of an adult male was also provided at a resolution of 256 x 256 x 256. A demo rendering program was ported to run on a Linux system along with a full import pipeline to load data into Houdini. This allowed direct comparison of the quality of US data with CT data.

[00116] With a basic conversion pipeline, filtering and rendering were performed on the two main data sets: the CT scan from Siemens and the US scan.

[00117] The following procedure was used:

[00118] First, raw data is imported into Houdini using the custom decoder described above for either the Siemens or GE .vol files. From here we have

a standard volume representation that can be processed using a variety of tools.

[00119] The following steps were used to produce renderable geometry:

a. Filter the raw volume with a mean value (box) filter.
b. Convert the volume into a level set.
c. Carve out obvious noise and outliers using a spherical volume cutter.
d. Smooth the level set with a Gaussian filter.
e. Convert the level set to polygons (renderable geometry).

[00120] From here three slightly different meshes are produced for blending:

a. Regular: the direct result of level set to polygon conversion.
b. Medium geometric smoothing: average neighboring point positions.
c. Aggressive geometric smoothing: average positions within a specific search radius.

[00121] The three meshes are blended with weights. The smoothest has the biggest influence, while the regular mesh has the least. The goal is to retain subtle variations in the data while maintaining the smooth shape. The resulting blended mesh is smoothed again at the edges, and finally we apply a “peaking” filter by pulling the mesh a tiny bit inward along the normal direction.

[00122] For rendering, Houdini's marble shader was used, which implements physically based subsurface scattering. This mimics the transport of light in semi-translucent materials like skin. Rendering produces several image layers that can be composited together and placed on top of the background..

[00123] Once imported, the same processing steps can be applied to any type of volume data to produce images. Although this set of steps requires some manual work, the process is repeatable and can be captured in a procedure within Houdini so that additional images can be produced relatively quickly.

[00124] FIG. 10A and FIG. 10B show a raw scan and isosurface render,

respectively, using CT data from the Siemens example data sets and FIG. 10C and FIG. 10D show a raw scan and isosurface render, respectively, for the GE US data. These were imported into Houdini using our custom decoder steps detailed above and then processed/rendered using Houdini native tools. Note that these data sets use very different imaging technology (CT versus US), but they are recorded at comparable resolution (albeit with the caveat mentioned above that the CT dataset has a resolution of 256 x 256 x 256). With the CT scan you can easily pick out many fine scale details of the subject including skin wrinkles and other fine geometric details. This is without any further processing of the raw volume data. US data seems inherently noisy.

[00125] FIG. 11A through FIG. 11C show a flow diagram for a processing method 300 for automatic high-quality rendering of arbitrary human dicom scans with a virtual camera. Method 300 enables input from three separate sources: dicom images 302 (e.g. arbitrary image scans from MRI, CT, etc.), 3D "standard" human library 304 (e.g. complete human body dataset (mesh models, textures, shaders, etc.)), and camera input 306 (arbitrary virtual camera view).

[00126] DICOM image data input 302 is fed into the automatic mesh generation module 310, wherein it reads the dicom images and generates the series of slices at first step or node 312. This step allows the user to optionally display the series of slices in the system viewport at 322a (i.e. hardware rendering).

[00127] Next at step 314 the series of slices are converted to a volume. This step allows the user to optionally display the generated volume in the system viewport at 322b.

[00128] Next at step 316, the volume data is clipped and filtered. Ideally, mean/Gaussian kernels are used for filtering. This step allows the user to optionally display the volume result in the system viewport at 322c.

[00129] Next at step 318, the isosurface is generated from the volume. Ideally, OTSU volume histogram thresholds are used to generate the isosurface. This step allows the user to optionally display the generated

isosurface in the system viewport at 322d.

[00130] Next at step 320, the polygon mesh shape is generated from the volume. This step allows the user to optionally display the generated polygon mesh shape in the system viewport at 322e. This output is then fed into the automatic mesh processing module 330.

[00131] At step 332, the generated mesh is analyzed and identified. The generated mesh is morphologically compared against all the ones of the standard human library in order to find its matching, and thus identify it. Heat equation/ propagation/ laplace-beltrami operator/ temperature distribution histograms may be used for the morphological/ shape analysis and matching algorithm. This step allows the user to optionally display the matching result in the system viewport at 322f.

[00132] At step 334, the generated mesh is aligned (e.g. translated, rotated and scaled in order to fit with its matching). An iterative closed-points algorithm is used for the alignment algorithm. This step allows the user to optionally display the fitting result in the system viewport at 322g.

[00133] At step 336, the generated mesh is reconstructed. An iterative closed-points algorithm is used for the reconstruction algorithm. The generated mesh is reconstructed by copying the missing parts from its matching. This step allows the user to optionally display the reconstruction result in system viewport at 322h.

[00134] At step 338, the generated mesh is texture mapped by copying the coordinates and assigned textures from its matching. An iterative closed-points algorithm is used for the mapping algorithm. This step allows the user to optionally display the texture mapping result in the system viewport at 322i.

[00135] At step 340, the rendering is output. The resulting mesh is software rendered at an arbitrary camera view, an arbitrary resolution, and with a high-quality texturing and shading.

[00136] Exemplary software code for carrying out the processing steps of method 300 is found in Table 1, below. Table 1 provides an embodiment of instructions contained in application programming 104 the may be

executable on a processor 106 to perform the functions shown in method 300 of FIG. 11, or any other method described herein.

[00137] F. Additional Applications

[00138] The animation of a laparoscopic appendectomy will demonstrate that the patient is in the supine position with the left arm tucked. The monitors are positioned at two o'clock and four o'clock and the patient is under general anesthesia. An incision is made in the midline of the umbilicus and the umbilical ring is dilated. A 12-mm blunt trochar is placed into the abdominal cavity which is then insufflated with CO_2 gas. Two 5-mm ports are placed under direct camera vision in the left flank. The operation then proceeds in conjunction with the animated CT image of the inflamed appendix as well as the animated simulation that has been developed from patient's CT scan. An exploration is performed which reveals all normal organs within the abdominal cavity except for the inflamed appendix. The adhesions are taken down using blunt and sharp dissection. The base of the appendix is lifted cephalad using a grasper. The mesoappendix is identified. The base of the appendix is then gently dissected using a Kittner dissector. Once a window is obtained between the base of the appendix and the mesoappendix a 12-mm stapler is placed into the abdominal cavity using the umbilical port. A 5-mm camera is used from the 5 mm lateral port. The base of the appendix is then stapled. The meso appendix is identified and stapled using a GIA stapling device with a vascular load. The appendix is then placed into a specimen bag and removed out of the umbilical port. The staple lines are examined. If irrigation is needed, it is performed at this point. The trochars are removed, the gas is removed, and the umbilical trochar site is closed with interrupted sutures of #0 Vicryl. The skin is closed with #4-0 Monocryl. Steri-Strips are applied. Sterile dressings are applied. The patient is awakened and then taken to the recovery room.

[00139] The animated simulation in accordance with the present description is used on the screen utilizing picture-in-a-picture technology. The animated simulation of the appendectomy is also used to help guide the

surgeon through the operation. The videos of actual laparoscopic appendectomies are linked to CT scans, MRI scans and ultrasounds will be used to develop the animated simulation. This constitutes personalized surgery/interventions based upon the patient's imaging studies and animated imaging. Similar personalized animated simulations can be developed for all operations and interventional procedures.

[00140] Personalized imaging using whole body scanning sequentially over time may also be implemented utilizing the systems and methods of the present description. The animated image and simulation methods may be used to create whole body scanning that to follow patients sequentially over time to understand their current health status as well as pathophysiology and the progression of any disease that might be present, such as cancer. Personalized imaging using whole body scanning may also be used to develop home imaging systems to develop weight loss, weight gain, general health conditions at home, and can be connected to an electronic medical record. For example, the patient's differential of muscle and fat versus bone can all be ascertained using a home imaging system using animated imaging.

[00141] The systems and methods of the present description may also be implemented for application of personal grooming, including facial make-up, other cosmetic applications, dress, etc in 3D for a number of special events

[00142] The systems and methods of the present description may also be combined with 3D printing technology to produce models of the patient's body, head, organs, cells for enhanced imaging, diagnoses, therapy and personal uses.

[00143] Personalized animated imaging for diagnostics (Embodiment 1) produces a photorealism rendering of the patient's body and organs. An actual color photograph of any diagnostic image transforms diagnostic capability for the radiologist and clinicians of all specialties, as well as education of the patient. The images may be used for real time simulations for practice of a procedure, as well as realtime use during the procedure, for interventionalists of all specialties, including all surgical specialties,

radiology, pulmonary, anesthesia, gastroenterology, etc. The images may be used on a daily basis to transform imaging, diagnostic capabilites and therapy in all hospitals and clinics worldwide. The images and animations are transformative for training in all specialties. The images may be used for home health systems to help patients monitor their weight, body mass, body fat content, body water content, body muscle content, which can be connected to an electronic medical record. The system would allow users to visualize and save images of their body images over time.

[00144] Personalized animated simulation for any interventional procedures, as provided in Embodiment 2, may be implemented to assist surgeons and interventionalists of all specialties in practice for a given procedure, and also help guide the interventionalist in real time through the procedure using artificial intelligence, like a GPS, and to track milestones of any procedure, as well as tracking the progress of the milestones for the procedure in real time in the medical records. This would allow standardization of procedures on a global scale and improve outcomes and quality of care and improve documentation of all procedures worldwide. This would also avoid the need for costly and inaccurate dictations and improve documentation of healthcare.

[00145] The systems and methods of the present description may also be implemented for education for patients, students, medical students, trainees and practicing physicians (CME).

[00146] In one embodiment, personalized imaging using whole body scanning may be performed sequentially over time, and may be coupled with artificial intelligence.

[00147] In another embodiment, the system may comprise a home animation imaging system to determine weight loss, weight gain, general health at home, fat content, muscle mass, water content, connected to an EMR using artificial intelligence to guide the patients through the process. The home animation imaging system would help patients recover from any procedure and also help guide the patient in real time through the recovery process using artificial intelligence to track milestones of the recovery, as well as

tracking the progress of the milestones for the recovery in real time in the medical records. This would lead to standardization of recovery from procedures on a global scale and improve outcomes, quality of care and quality of life. This would also avoid the need for costly readmissions and improve documentation of home healthcare.

[00148] The technology of the present description may be implemented to evaluate employees worldwide, including the evaluation of athletes. The technology may be used to evaluate both anatomy and physiology (form and function).

[00149] Another implementation of technology is 3D printing to create models of the patients' organs, patients head and torso, and diseases.

[00150] The technology may comprise a system configured for personal grooming, including facial make-up, other cosmetic applications, dress, etc., in 3D for any number of special events, such as evening socials, proms, weddings, bar mitzvahs, etc.

[00151] Face and body recognition technology may be implemented, with populating the site with internal organs. Medical knowledge may be used to navigate the animated imaging and videos, and be used for movies, television, music videos and internet as well for research purposes in laboratories, clinics, hospitals and medical schools.

[00152] SIMaging systems and methods may be used for military, paramedic and hospital emergency room rescues. The system may be incorporated into a portable unit, like an ultrasound configured to take images/photos of the internal organs of an injured soldier or civilian and relay that information back to a MASH unit for guidance based upon the image/photo of the internal injury.

[00153] In the hospital, the CT scans and ultrasounds are greatly enhanced using the system of the present description, showing images of the internal organs, thus more clearly defining the injuries of the patient. Artificial intelligence, in the form of a virtual paramedic, may be configured into the system to guide the military medic, paramedic, emergency physician and/or trauma surgeon in the care of the injured soldier or civilian. The actual care

can be applied by a robot, which is guided by the virtual paramedic or virtual surgeon. The simulations would also be used for education for all health care providers, guided by the virtual paramedic or virtual surgeon.

[00154] Mobile CT, MRI and ultrasound mobile units may be configured to make house calls to patients for imaging and care conveyed back to the electronic medical record of the health system. The actual images and care can be applied by a robot, which is guided by the virtual nurse, virtual paramedic or virtual surgeon and documented in real time. The robot could be a personal avatar that assists with all applications of SIMaging to help with health maintenance for each person.

[00155] SIMaging software in accordance with the present description may be used in home portable units, such as ultrasounds, that are safe and easy to use to generate photorealistic images of the internal organs. The actual images can be taken by a robot avatar, which is guided by artificial intelligence in the form of the virtual paramedic or virtual nurse. Therefore, SIMaging software could be part of robotic avatars that are used at home as virtual trainers, valets, lady's maids, paramedics or virtual nurses.

[00156] SIMaging software in accordance with the present description may be configured for automatic photographic enhancement, in which the SIMaging software is implemented in a camera to enhance the quality of photograph produced, such as that seen with common manual enhancement of images via "airbrushing" or the like. The photographs can be taken by the person or a robotic avatar, and are guided by the virtual photographic SIMaging software.

[00157] SIMaging software configured for personal grooming software may include tutorials on diets, weight loss, weight gain, exercise programs, attire, facial make-up, other cosmetic applications, etc., in 3D high definition. This may include artificial intelligence, in the form of a virtual valet or lady's maid or personal avatar, to help the viewer with tutorials on the basic principles of how to eat, exercise, to dress and how to apply make-up to optimize one's health and image. The SIMaging personal grooming software may incorporate how the world's experts in diet,

exercise, fashion design and make-up apply their craft for any given event world wide to advise the viewer on how to prepare, dress and apply their make-up for any number of special events, such as vacations, evening socials, red carpet events, balls, proms, weddings, bar mitzvahs, funerals, on a world wide basis. The make-up can be applied by a robotic avatar, which is guided by SIMaging software. The software will include how the stars of all entertainment fields dress and apply for make-up for any given social event on a global basis.

[00158] An interested viewer may use SIMaging personal grooming software to learn how to dress and apply make-up on their own high definition images using artificial intelligence, as a virtual valet or virtual lady's maid or personal avatar, to take them through the basics of make-up application. The viewer will get to see how they look with different styles of clothing. The viewer will get to see how they appear with each different application of eye liner, powder, facial liners, rouge, lipstick, eyebrow enhancement or trimming, wigs, false eyelashes, skin color, tanning, etc. The make-up can be applied by a robotic avatar, which is guided by SIMaging software.

[00159] The SIMaging personal grooming software may be used for educational and training purposes in beauty and design schools and acting schools worldwide, guided by the virtual valet or lady's maid or personal avatar.

[00160] The SIMaging personal grooming software may be used by entertainers of all types, producers, directors, dress designers worldwide to help them prepare for any given entertainment event, such as movies, television, stage, ballet, shows, concerts, sporting events, etc., all guided by the virtual valet or lady's maid or personal avatar. The software will demonstrate their images, or images of a given performer or athlete, in 3D high definition, in any given attire and make-up in helping to prepare for any given entertainment event. For example, producers, directors and casting agents will use this software to help select to most appropriate actor or actress for a role in a propose movie or television show, as well as help the actor or actress prepare for that movie or show. The make-up can be

applied by a robotic avatar, which is guided by the SIMaging software.

[00161] The SIMaging personal grooming software may assist in teaching the basics of how exercise and diet will help them to gain or lose weight and how fit they will appear with selected exercise regimens, all guided by the virtual trainer, virtual valet or virtual lady's maid or personal avatar, which is guided by SIMaging software.

[00162] Once the basics are learned, the viewer may use the software and artificial intelligence, in the form of the virtual valet or virtual lady's maid or personal avatar, to help prepare for any given social event on a worldwide basis. For example, if the person attends a wedding in China, the SIMaging software may help them prepare the appropriate style of dress and make-up for such an event and allow how they will appear for that event. The make-up can be applied by a robotic avatar, which is guided by SIMaging software. The software will help them keep track of their own clothing and what might be needed for any given event, as well as what clothes and make-up that were chosen for any previous event. The SIMaging software and artificial intelligence, in the form of the virtual valet or virtual lady's maid or personal avatar, will give advice as to what the world fashion designers and make-up artists would choose for any given event and how the stars might appear for that event or for any previous event. The clothing can be retrieved from the closet by a robot, which is guided by the virtual valet or virtual lady's maid or personal avatar.

[00163] The SIMaging personal grooming software may be configured to demonstrate in 3D high definition how the viewer will appear if they chose an exercise routine and diet for any given event, all guided by the virtual trainer, virtual valet or lady's maid or personal avatar. For example, if the person were planning a beach vacation, the SIMaging software will show the viewer in high definition how their body will look if they were to lose five or ten pounds on a given diet as well as an exercise routine. Routines and diets of leading experts and trainers, as well as predicted results, will be part of the software.

[00164] The SIMaging personal grooming software may be used by

healthcare providers of all specialties for educational purposes to show patients how they would appear after any body altering therapy, such as plastic surgery, morbid obesity surgery, any operation on the outer body, as well as after chemotherapy, steroids, all guided by the virtual professor, virtual doctor, virtual valet, virtual lady's maid or personal avatar. The software may be used to educate patients on how they and their internal organs will appear with harmful behavioral habits, such as smoking and other illicit drug use, as well as over eating or under eating. The software can also be used to educate patients on how they and their internal organs will deteriorate due to the process of any given disease and how therapies might alter the deterioration of appearance.

[00165] The SIMaging personal grooming software may be used by the individual and/or healthcare providers of all specialties for educational purposes to show the viewer how they and their internal body composition will age over time, all guided by the virtual doctor, virtual valet, virtual lady's maid or personal avatar.

[00166] The SIMaging personal grooming software may be used by governments, national security agencies, military agencies, and police forces of all specialties for educational purposes to demonstrate how their personnel will appear after training, make-up, dress, or any body altering procedures for the purposes of national security, all guided by artificial intelligence in the form of the virtual tutor, virtual valet, lady's maid or personal avatar. The make-up can be applied by a robot avatar, which is guided by the SIMaging software.

[00167] Conversely, the SIMaging personal grooming software may be used by governments, national security agencies, military agencies and police forces of all specialties for identification purposes to demonstrate how any criminal might appear after make-up, dress, or any body altering procedures for the purposes of national security, all guided by artificial intelligence in the form of the virtual agent, virtual tutor, virtual professor, virtual valet, lady's maid or personal avatar.

[00168] The SIMaging software may be configured in the form of computer

games that assist in learning/practicing medical and surgical interventions, to practice interventions/operations in real time on actual patient images.

[00169] The SIMaging software may be configured in the form of home health systems connected to electronic medical records.

[00170] The SIMaging software may be configured for real time simulations that could be used for practice of any intervention/operation, as well as realtime use during the procedure, for interventionalists of all specialties, including all surgical specialties, gastroenterology, radiology, pulmonary, anesthesia, pain medicine, cardiology, etc.

[00171] The SIMaging software may be coupled with artificial intelligence to guide the surgeon/interventionalist through procedures, like a GPS system, in real time, while documenting each step of the procedure according to standard operating protocols, thus standardizing and documenting procedures and entering the information into databases.

[00172] The systems and methods of the present description improve quality of care, improve documentation and lower costs of complications, readmissions and documentation.

[00173] The SIMaging software may be configured as home health systems that would help patients monitor their weight, body mass, body fat content, body water content, body muscle content, which can be connected to an electronic medical record.

[00174] The images generated by SIMaging software may be used for entertainment purposes; examples would be (a) to produce an animated photograph or movie of a baby in utero for parents expecting their developing baby; (b) to develop medical video games using simulations and for movies.

[00175] The SIMaging software may be configured as a home animation imaging system to determine weight loss, weight gain, general health at home, fat content, muscle mass, water content, connected to an EMR using artificial intelligence to guide the patients through the process.

[00176] The SIMaging software may be configured as a home animation imaging system to help patients recover from any procedure and also help

guide the patient in real time through the recovery process using artificial intelligence to track milestones of the recovery, as well as tracking the progress of the milestones for the recovery in real time in the medical records.

[00177] The SIMaging software may be configured as gaming systems that involve the human body, repair of the human body, and portraying any injury of the human body such as gunshots, stabbings, car crashes, and other trauma that are currently used in gaming systems.

[00178] The SIMaging software may be configured to create realistic simulations of trauma to the body and provide the ability to repair the trauma.

[00179] The SIMaging software may be configured to use of artificial intelligence in the form of personal avatars, virtual doctors, virtual surgeons, virtual professors, virtual paramedic, virtual valet, virtual lady's maid, etc to guide the viewer through the simulation, and the use of robots, which will actually implement the numerous applications. For example, personalized robotic avatars will actually perform the home healthcare, operations, procedures, trauma rescues in the field, make-up application, clothing, photography, etc. for all SIMaging applications, guided by the artificial intelligence built into the software. The robots can also be guided by the doctors or paramedics in real time. Each person could have their own personal avatar, either virtual or an actual robot, which helps guide them through their own health maintenance or personal grooming using SIMaging software.

[00180] Embodiments of the technology of this disclosure may be described with reference to flowchart illustrations of methods and systems according to embodiments of the technology, and/or algorithms, formulae, or other computational depictions, which may also be implemented as computer program products. In this regard, each block or step of a flowchart, and combinations of blocks (and/or steps) in a flowchart, algorithm, formula, or computational depiction can be implemented by various means, such as hardware, firmware, and/or software including one or more computer

program instructions embodied in computer-readable program code logic. As will be appreciated, any such computer program instructions may be loaded onto a computer, including without limitation a general purpose computer or special purpose computer, or other programmable processing apparatus to produce a machine, such that the computer program instructions which execute on the computer or other programmable processing apparatus create means for implementing the functions specified in the block(s) of the flowchart(s).

[00181] Accordingly, blocks of the flowcharts, algorithms, formulae, or computational depictions support combinations of means for performing the specified functions, combinations of steps for performing the specified functions, and computer program instructions, such as embodied in computer-readable program code logic means, for performing the specified functions. It will also be understood that each block of the flowchart illustrations, algorithms, formulae, or computational depictions and combinations thereof described herein, can be implemented by special purpose hardware-based computer systems which perform the specified functions or steps, or combinations of special purpose hardware and computer-readable program code logic means.

[00182] Furthermore, these computer program instructions, such as embodied in computer-readable program code logic, may also be stored in a computer-readable memory that can direct a computer or other programmable processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory produce an article of manufacture including instruction means which implement the function specified in the block(s) of the flowchart(s). The computer program instructions may also be loaded onto a computer or other programmable processing apparatus to cause a series of operational steps to be performed on the computer or other programmable processing apparatus to produce a computer-implemented process such that the instructions which execute on the computer or other programmable processing apparatus provide steps for implementing the functions specified in the block(s) of the

flowchart(s), algorithm(s), formula(e), or computational depiction(s).

[00183] From the discussion above it will be appreciated that the technology described herein can be embodied in various ways, including but not limited to the following:

[00184] 1. A computer implemented method for enhanced imaging, the method comprising: (a) transforming a non-color, non-photorealistic image into a high definition colorized photorealistic image; (b) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00185] 2. A computer implemented method for enhanced imaging, the method comprising: (a) transforming a non-color, two-dimensional image generated from a diagnostic imaging device into high definition colorized, photorealistic image; (b) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00186] 3. The method of any preceding embodiment, further comprising creating animated simulations based on a plurality of said photorealistic images.

[00187] 4. The method of claim any preceding embodiment, further comprising highlighting an area of interest in the photorealistic image using a molecular contrast promoter.

[00188] 5. The method of any preceding embodiment, further comprising automatically generating a diagnosis by evaluating characteristics of the areas of interest in the photorealistic image.

[00189] 6. The method of any preceding embodiment, further comprising: using functional genomics and molecular imaging to generate a molecular contrast; using the molecular contrast to highlight the area of interest.

[00190] 7. The method of any preceding embodiment, further comprising, further comprising automatically generating a diagnosis by evaluating characteristics of the areas of interest in the photorealistic image.

[00191] 8. A computer implemented method for creating an animated

simulation, the method comprising: (a) transforming a plurality of images of a biological component, feature, characteristic, assembly, or structure, or a combination thereof, into high definition colorized photorealistic images; and (b) assembling said photorealistic images into an animated simulation; (c) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00192] 9. An apparatus for enhanced imaging, the apparatus comprising: (a) a computer processor; and (b) programming in a non-transitory computer readable medium and executable on the computer processor for transforming a non-color, non-photorealistic image into a high definition colorized photorealistic image.

[00193] 10. An apparatus for enhanced imaging, the apparatus comprising: (a) a computer processor; and (b) programming in a non-transitory computer readable medium and executable on the computer processor for transforming a non-color, two-dimensional image generated from a diagnostic imaging device into high definition colorized, photorealistic image.

[00194] 12. The apparatus of any preceding embodiment, wherein said programming is configured to create animated simulations based on a plurality of said photorealistic images.

[00195] 13. The apparatus of any preceding embodiment, wherein said programming is configured to highlight an area of interest in the photorealistic image using a molecular contrast promoter.

[00196] 14. The apparatus of any preceding embodiment, wherein said programming is configured for automatically generating a diagnosis by evaluating characteristics of the areas of interest in the photorealistic image.

[00197] 15. The apparatus of any preceding embodiment, wherein said programming is configured for performing steps comprising: using functional genomics and molecular imaging to generate a molecular contrast; and using the molecular contrast to highlight the area of interest.

[00198] 16. The apparatus of any preceding embodiment, wherein said programming is configured for automatically generating a diagnosis by evaluating characteristics of the areas of interest in the photorealistic image.

[00199] 17. An apparatus for creating an animated simulation, the apparatus comprising:(a) a computer processor; and (b) programming in a non-transitory computer readable medium and executable on the computer processor for: (i) transforming a plurality of images of a biological component, feature, characteristic, assembly, or structure, or a combination thereof, into high definition colorized photorealistic images; and (ii) assembling said photorealistic images into an animated simulation.

[00200] 18. An enhanced image, comprising: (a) a high definition colorized photorealistic image transformed from a non-color, non-photorealistic image; (b) wherein image transformation is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00201] 19. An enhanced image, comprising: (a) a high definition colorized photorealistic image transformed from a non-color, two-dimensional image generated from a diagnostic imaging device; (b) wherein image transformation is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00202] 20. A animated simulation, comprising: (a) an assembly of a high definition colorized photorealistic images transformed from non-color, non-photorealistic images; (b) wherein image transformation and assembly is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00203] 21. A animated simulation, comprising: (a) an assembly of a high definition colorized photorealistic images transformed from a non-color, two-dimensional image generated from a diagnostic imaging device; (b) wherein image transformation and assembly is performed by executing

programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

[00204] 22. An apparatus for enhanced imaging, the apparatus comprising: (a) a computer processor; and (b)a non-transitory computer-readable memory storing instructions executable by the computer processor; (c) wherein said instructions, when executed by the computer processor, perform steps comprising: (i)generating a database of parametric anatomy comprising one or more of volume data and isometric surface models of one or more aspects of the anatomy; (ii)tagging one or more objects within the parametric anatomy; (iii) inputting patient data comprising an imaging scan of a target patient anatomy of a patient; (iv) configuring a base parametric model of patient anatomy as a function of input patient data comprising one or more physical characteristics of the patient; (v) applying data relating to the imaging scan to the base parametric model; (vi) searching the data relating to the imaging scan for one or more markers within the data; (vii) aligning the parametric model to the one or more markers of the imaging scan; and (viii) rendering the aligned parametrical model and imaging scan for photo-realistic display of the patient target anatomy.

[00205] 23. The apparatus of any preceding embodiment, wherein data relating to the imaging scan comprises DICOM data from one or more of an MRI, CT, or ultrasound scan of the patient.

[00206] 24. The apparatus of any preceding embodiment, wherein the database is generated by acquiring input from patient data comprising one or more of patient scans, statistics or photos relating to patient.

[00207] 25. The apparatus of any preceding embodiment, wherein the isometric surface models are configured for photo-real real-time VR rendering.

[00208] 26. The apparatus of any preceding embodiment, wherein the tagged data is configured so that that can be turned on or off for viewing.

[00209] 27. The apparatus of any preceding embodiment, the instructions further comprising: allowing manual input for selecting the one or more

markers.

[00210] 28. The apparatus of any preceding embodiment, wherein aligning the parametric model to the one or more markers of the imaging scan comprises isometric surface alignment of the parametric anatomical geometry to the patient's DICOM scan data.

[00211] 29. The apparatus of any preceding embodiment, wherein the alignment is done taking into account input patient data relating to one or more of weight, height, BMI, X-ray, and patient photos.

[00212] 30 The apparatus of any preceding embodiment, wherein the alignment is performed on both volume data and isometric surfaces.

[00213] 31. The apparatus of any preceding embodiment, wherein the alignment is adjusted by manual input via selecting one or more structures on slices of the MRI or CT scan to fine-tune the base parametric model.

[00214] 32. The apparatus of any preceding embodiment, the instructions further configured for: applying photographic reference of skin color to the output parametric model.

[00215] 33. The apparatus of any preceding embodiment, the instructions further configured for: auto-alignment and projection of one or more of the following to the parametric model: patient wounds, surgeon markups, X-rays, notes and other text files.

[00216] 34. A computer implemented method for enhanced imaging, the method comprising: generating a database of parametric anatomy comprising one or more of volume data and isometric surface models of one or more aspects of the anatomy; tagging one or more objects within the parametric anatomy; inputting patient data comprising an imaging scan of a target patient anatomy of a patient; configuring a base parametric model of patient anatomy as a function of input patient data comprising one or more physical characteristics of the patient; applying data relating to the imaging scan to the base parametric model; searching the data relating to the imaging scan for one or more markers within the data; aligning the parametric model to the one or more markers of the imaging scan; and rendering the aligned parametrical model and imaging scan for photo-

realistic display of the patient target anatomy.

[00217] 35. The method of any preceding embodiment, wherein data relating to the imaging scan comprises DICOM data from one or more of an MRI, CT, or ultrasound scan of the patient.

[00218] 36. The method of any preceding embodiment, wherein the database is generated by acquiring input from patient data comprising one or more of patient scans, statistics or photos relating to patient.

[00219] 37. The method of any preceding embodiment, wherein the isometric surface models are configured for photo-real real-time VR rendering.

[00220] 38. The method of any preceding embodiment, wherein the tagged data is configured so that that can be turned on or off for viewing.

[00221] 39. The method of any preceding embodiment the method further comprising: allowing manual input for selecting the one or more markers.

[00222] 40. The method of any preceding embodiment, wherein aligning the parametric model to the one or more markers of the imaging scan comprises isometric surface alignment of the parametric anatomical geometry to the patient's DICOM scan data.

[00223] 41. The method of any preceding embodiment, wherein the alignment is done taking into account input patient data relating to one or more of weight, height, BMI, X-ray, and patient photos.

[00224] 42 The method of any preceding embodiment, wherein the alignment is performed on both volume data and isometric surfaces.

[00225] 43. The method of any preceding embodiment, wherein the alignment is adjusted by manual input via selecting one or more structures on slices of the MRI or CT scan to fine-tune the base parametric model.

[00226] 44. The method of any preceding embodiment, the method further comprising: applying photographic reference of skin color to the output parametric model.

[00227] 45. The method of any preceding embodiment, the method further comprising: auto-alignment and projection of one or more of the following to the parametric model: patient wounds, surgeon markups, X-rays, notes and other text files.

[00228] Although the description herein contains many details, these should not be construed as limiting the scope of the disclosure but as merely providing illustrations of some of the presently preferred embodiments. Therefore, it will be appreciated that the scope of the disclosure fully encompasses other embodiments which may become obvious to those skilled in the art.

[00229] In the claims, reference to an element in the singular is not intended to mean "one and only one" unless explicitly so stated, but rather "one or more." All structural, chemical, and functional equivalents to the elements of the disclosed embodiments that are known to those of ordinary skill in the art are expressly incorporated herein by reference and are intended to be encompassed by the present claims. Furthermore, no element, component, or method step in the present disclosure is intended to be dedicated to the public regardless of whether the element, component, or method step is explicitly recited in the claims. No claim element herein is to be construed as a "means plus function" element unless the element is expressly recited using the phrase "means for". No claim element herein is to be construed as a "step plus function" element unless the element is expressly recited using the phrase "step for".

Table 1

Firmware

```
////////////////////////////////////////////////////////////////////////
//
// HDX_DCMseriesReader (Maya node)
//
// reads the DICOM images and generates the series of slices.
// this node allows to optionally display the series of slices in the maya
// viewport (hardware rendering)
//
////////////////////////////////////////////////////////////////////////


// Maya includes
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MFnPlugin.h>
#include <maya/MMaterial.h>
#include <maya/MSelectionList.h>
#include <maya/MSelectionMask.h>
#include <maya/MDrawData.h>
#include <maya/MMatrix.h>
#include <maya/MObjectArray.h>
#include <maya/MDagPath.h>

// STL includes
#include <limits>

// GDCM Includes
#include <gdcmImageReader.h>
#include <gdcmImageHelper.h>
#include <gdcmRescaler.h>
#include <gdcmAttribute.h>
#include <gdcmUnpacker12Bits.h>

// Core includes
#include <Core/DataBlock/StdDataBlock.h>


////////////////////////////////////////////////////////////////////////
// HDX_DCMseriesReader
////////////////////////////////////////////////////////////////////////

MTypeId HDX_DCMseriesReader::id( 0x8000001 );
```

```
void* HDX_DCMseriesReader::creator()
{
        return new HDX_DCMseriesReader();
}

MStatus HDX_DCMseriesReader::initialize()
{
        return MS::kSuccess;
}

class HDX_DCMseriesReader
{
        public:

        virtual ~HDX_DCMseriesReader() {;}
                        HDX_DCMseriesReader();

        public:

                HDX_DCMseriesReader() :
                rescale_slope_( 0.0 ),
                rescale_intercept_( 0.0 ),
                buffer_length_( 0 ),
                slice_data_size_( 0 ),
                pixel_type_( Core::DataType::UCHAR_E ),
                origin_( 0.0, 0.0, 0.0 ),
                row_direction_( 1.0, 0.0, 0.0 ),
                col_direction_( 0.0, 1.0, 0.0 ),77
                slice_direction_( 0.0, 0.0, 1.0 ),
                x_spacing_( 1.0 ),
                y_spacing_( 1.0 ),
                z_spacing_( 1.0 ),
                read_header_( false ),
                swap_xy_spacing_( false )
        {
        }

        // READ_HEADER
        // Read the header of the dicom file and fill out its information into this
private class
        bool read_header();

        // READ_DATA
        // Read the dicom data into this private class
        bool read_data();
```

```
        // READ_IMAGE
        bool read_image( const std::string& filename, char* buffer );

        public:
        // Pointer to interface class
        HDX_DCMseriesReader* reader_;

        double rescale_slope_;
        double rescale_intercept_;
        unsigned long buffer_length_;
        unsigned long slice_data_size_;

        Core::GridTransform grid_transform_;
        Core::DataType pixel_type_;

        Core::Point origin_;
        Core::Vector row_direction_;
        Core::Vector col_direction_;
        Core::Vector slice_direction_;

        double x_spacing_;
        double y_spacing_;
        double z_spacing_;

        // Data block with actual data (generated by read_data)
        Core::DataBlockHandle data_block_;

        // Meta data (generated by read_data)
        nodeMetaData meta_data_;

        // Whether the header of the files has been parsed
        bool read_header_;

        // Whether to swap xy spacing
        bool swap_xy_spacing_;

        public:

        void draw_slice( nodeSceneItemHandle node_item, const Core::Matrix&
proj_mat,
                ProxyRectangleHandle rect = ProxyRectangleHandle() );
        void map_slice_texture(Core::Texture2DHandle slice_tex, int width, int
height,
                double left, double right, double bottom, double top,
                const Core::Matrix& proj_mat,
                ProxyRectangleHandle rect);
        void map_large_slice_texture(Core::Texture2DHandle slice_tex, int width,
```

```
int height,
                double left, double right, double bottom, double top,
                const Core::Matrix& proj_mat,
                ProxyRectangleHandle rect);
};

////////////////////////////////////////////////////////////////////////

bool HDX_DCMseriesReader::read_header()
{
  if ( this->read_header_ ) return true;

  // Get the filenames
  std::vector<std::string> filenames = this->reader_->get_filenames();
  if ( filenames.size() == 0 )
  {
    this->reader_->print_error( "no files were specified." );
    return false;
  }

  // Setup some compatibility rules for GDCM
  gdcm::ImageHelper::SetForcePixelSpacing( true );
  gdcm::ImageHelper::SetForceRescaleInterceptSlope( true );

  // Read the first file and extract information out of it;

  gdcm::ImageReader reader;
  reader.SetFileName( filenames[ 0 ].c_str() );
  if ( !reader.Read() )
  {
    this->reader_->print_error( std::string( "cannot read file '" ) + filenames[ 0 ]+ "'."
);
    return false;
  }

  const gdcm::Image& image = reader.GetImage();
  const gdcm::File& file = reader.GetFile();
  const gdcm::DataSet& ds = file.GetDataSet();
  const unsigned int* dims = image.GetDimensions();
  const gdcm::PixelFormat& pixeltype = image.GetPixelFormat();
  this->buffer_length_ = image.GetBufferLength();

  if ( pixeltype.GetSamplesPerPixel() != 1 )
  {
    this->reader_->print_error( "unsupported pixel format." );
    return false;
  }
```

```
unsigned int num_of_dimensions = image.GetNumberOfDimensions();
if ( num_of_dimensions != 2 && num_of_dimensions != 3 )
{
  this->reader_->print_error( "unsupported number of dimensions." );
  return false;
}

this->grid_transform_.set_nx( dims[ 0 ] );
this->grid_transform_.set_ny( dims[ 1 ] );

if ( num_of_dimensions == 2 )
{
  this->grid_transform_.set_nz( filenames.size() );
}
else
{
  this->grid_transform_.set_nz( dims[ 2 ] );
}

this->rescale_intercept_ = image.GetIntercept();
this->rescale_slope_ = image.GetSlope();

gdcm::Rescaler rescaler;
rescaler.SetIntercept( this->rescale_intercept_ );
rescaler.SetSlope( this->rescale_slope_ );
rescaler.SetPixelFormat( pixeltype );
gdcm::PixelFormat::ScalarType output_pixel_type =
rescaler.ComputeInterceptSlopePixelType();

switch( output_pixel_type )
{
  case gdcm::PixelFormat::INT8:
    this->pixel_type_ = Core::DataType::CHAR_E;
    break;
  case gdcm::PixelFormat::UINT8:
    this->pixel_type_ = Core::DataType::UCHAR_E;
    break;
  case gdcm::PixelFormat::INT12:
    this->pixel_type_ = Core::DataType::SHORT_E;
    break;
  case gdcm::PixelFormat::UINT12:
    this->pixel_type_ = Core::DataType::USHORT_E;
    break;
  case gdcm::PixelFormat::INT16:
    this->pixel_type_ = Core::DataType::SHORT_E;
    break;
```

```
      case gdcm::PixelFormat::UINT16:
        this->pixel_type_ = Core::DataType::USHORT_E;
        break;
      case gdcm::PixelFormat::INT32:
        this->pixel_type_ = Core::DataType::INT_E;
        break;
      case gdcm::PixelFormat::UINT32:
        this->pixel_type_ = Core::DataType::UINT_E;
        break;
      case gdcm::PixelFormat::FLOAT32:
        this->pixel_type_ = Core::DataType::FLOAT_E;
        break;
      case gdcm::PixelFormat::FLOAT64:
        this->pixel_type_ = Core::DataType::DOUBLE_E;
        break;
      default:
        this->reader_->print_error( "encountered an unknown data type." );
        return false;
    }

    double epsilon = std::numeric_limits<double>::epsilon() * 10.0;

    this->slice_data_size_ = GetSizeDataType( this->pixel_type_ ) * dims[ 0 ] * dims[
  1 ];

    // Compute the grid transform
    const double* spacing = image.GetSpacing();
    const double* origin = image.GetOrigin();
    const double* dircos = image.GetDirectionCosines();

    this->row_direction_ = Core::Vector( dircos[ 0 ], dircos[ 1 ], dircos[ 2 ] );
    this->col_direction_ = Core::Vector( dircos[ 3 ], dircos[ 4 ], dircos[ 5 ] );
    this->slice_direction_ = Core::Cross( this->row_direction_, this->col_direction_ );
    this->slice_direction_.normalize();
    this->row_direction_.normalize();
    this->col_direction_.normalize();

    this->origin_[ 0 ] = origin[ 0 ];
    this->origin_[ 1 ] = origin[ 1 ];
    this->origin_[ 2 ] = origin[ 2 ];

    this->x_spacing_ = spacing[ 0 ];
    this->y_spacing_ = spacing[ 1 ];

    gdcm::Tag slice_thickness_tag( 0x0018,0x0050 );
    gdcm::Tag slice_distance_tag( 0x0018,0x0088 );
    gdcm::Tag patient_position_tag( 0x0020, 0x0032 );
```

```
bool found_thickness = false;

if( ds.FindDataElement( slice_thickness_tag ) ) // Slice Thickness
{
  const gdcm::DataElement& de = ds.GetDataElement( slice_thickness_tag );
  if ( ! de.IsEmpty() )
  {
    gdcm::Attribute< 0x0018, 0x0050 > slice_thickness;
    slice_thickness.SetFromDataElement( de );
    double thickness = slice_thickness.GetValue();
    if ( thickness > epsilon )
    {
      this->z_spacing_ = thickness;
      found_thickness = true;
    }
        else
        {
           this->reader_->print_warning( "encountered an incorrect value in the
slice thickness tag." );
        }
  }
}
else if( ds.FindDataElement( slice_distance_tag ) )
{
  const gdcm::DataElement& de = ds.GetDataElement( slice_distance_tag );
  if ( ! de.IsEmpty() )
  {
    gdcm::Attribute< 0x0018, 0x0088 > slice_thickness;
    slice_thickness.SetFromDataElement( de );
    double thickness = slice_thickness.GetValue();
    if ( thickness > epsilon )
    {
      this->z_spacing_ = thickness;
      found_thickness = true;
    }
        else
        {
           this->reader_->print_warning( "encountered an incorrect value in the
slice distance tag." );
        }
  }
}

if ( filenames.size() > 1 && ds.FindDataElement( patient_position_tag ) )
{
  gdcm::ImageReader reader2;
```

```
    reader2.SetFileName( filenames[ 1 ].c_str() );
    if ( !reader2.Read() )
    {
      this->reader_->print_error( "can't read file " + filenames[ 1 ] );
      return false;
    }

    const gdcm::Image &image2 = reader2.GetImage();

    const double* origin2 = image2.GetOrigin();
    Core::Vector origin_vec( origin[ 0 ], origin[ 1 ], origin[ 2 ] );
    Core::Vector origin_vec2( origin2[ 0 ], origin2[ 1 ], origin2[ 2 ] );
    Core::Vector dir = origin_vec2 - origin_vec;

    double spacing = dir.length();

      if ( found_thickness && Core::Abs( this->z_spacing_ - spacing ) > epsilon )
      {
        this->reader_->print_warning( "slice spacing in DICOM header is
inconsistent, using the patient slice location to derive the z spacing." );
      }

    if ( spacing < -epsilon || spacing > epsilon )
    {
      this->z_spacing_ = spacing;
      this->slice_direction_ = dir;
      this->slice_direction_.normalize();
    }
    else
    {
        this->reader_->print_warning( "slice spacing in DICOM header is too close
to zero, reseting it to 1.0." );
      this->z_spacing_ = 1.0;
    }
  }
  else if ( found_thickness == false )
  {
      this->reader_->print_warning( "no slice spacing is assigned in the DICOM
header." );
    this->z_spacing_ = 1.0;
  }

  this->grid_transform_.load_basis( this->origin_, this->row_direction_ * this-
>x_spacing_,
    this->col_direction_ * this->y_spacing_,
    this->slice_direction_ * this->z_spacing_ );
  this->grid_transform_.set_originally_node_centered( false );
```

```
  this->read_header_ = true;

  return true;
}

//////////////////////////////////////////////////////////////////////

bool HDX_DCMseriesReader::read_data()
{
  if ( this->swap_xy_spacing_ )
  {
    std::swap( this->x_spacing_, this->y_spacing_ );
  }

  this->grid_transform_.load_basis( this->origin_, this->row_direction_ * this->x_spacing_,
    this->col_direction_ * this->y_spacing_,
    this->slice_direction_ * this->z_spacing_ );

  this->grid_transform_.set_originally_node_centered( false );

  this->data_block_ = Core::StdDataBlock::New( this->grid_transform_, this->pixel_type_ );

  char* data = reinterpret_cast< char* >( this->data_block_->get_data() );
  std::vector<std::string> filenames = this->reader_->get_filenames();

  for ( size_t i = 0; i < filenames.size(); i++ )
  {
    if ( !this->read_image( filenames[ i ], data + this->slice_data_size_ * i ) )
    {
      this->data_block_.reset();
      return false;
    }
  }

  if ( filenames.size() )
  {
    InputFilesID inputfile_id = this->reader_->get_inputfiles_id();
    this->meta_data_.meta_data_ = Core::ExportToString( filenames ) + "|" +
      Core::ExportToString( inputfile_id );
    this->meta_data_.meta_data_info_ = "dicom_filename";
  }

  return true;
}
```

```
////////////////////////////////////////////////////////////////////////

bool HDX_DCMseriesReader::read_image( const std::string& filename, char*
buffer )
{
  gdcm::ImageReader reader;
  reader.SetFileName( filename.c_str() );

  if ( !reader.Read() )
  {
    this->reader_->print_error( "failed to read file '" + filename + "'" );
    return false;
  }

  gdcm::Image& image = reader.GetImage();
  if ( this->buffer_length_ != image.GetBufferLength() )
  {
    this->reader_->print_error( "images in the series have different sizes" );
    return false;
  }

  image.GetBuffer( buffer );
  const gdcm::PixelFormat& pixeltype = image.GetPixelFormat();
  if ( pixeltype == gdcm::PixelFormat::UINT12 )
  {
    if ( this->rescale_slope_ != 1.0 || this->rescale_intercept_ != 0.0 )
    {
      this->reader_->print_error( "unsupported data format" );
      return false;
    }

    std::vector< char > copy( this->buffer_length_ );
    memcpy( &copy[ 0 ], buffer, this->buffer_length_ );
    if ( !gdcm::Unpacker12Bits::Unpack( buffer, &copy[ 0 ], this->buffer_length_ ) )
    {
      this->reader_->print_error( "failed to unpack 12bit data" );
      return false;
    }
  }

  if ( this->rescale_slope_ != 1.0 || this->rescale_intercept_ != 0.0 )
  {
    gdcm::Rescaler rescaler;
    rescaler.SetIntercept( this->rescale_intercept_ );
    rescaler.SetSlope( this->rescale_slope_ );
    rescaler.SetPixelFormat( pixeltype );
```

```
    std::vector< char > copy( this->buffer_length_ );
    memcpy( &copy[ 0 ], buffer, this->buffer_length_ );
    rescaler.Rescale( buffer, &copy[ 0 ], this->buffer_length_ );
   }

  return true;
}

//////////////////////////////////////////////////////////////////////

void HDX_DCMseriesReader::set_dicom_swap_xyspacing_hint( bool
swap_xy_spacing )
{
  this->private_->swap_xy_spacing_ = swap_xy_spacing;
}

//////////////////////////////////////////////////////////////////////

bool HDX_DCMseriesReader::get_file_info( nodereaderFileInfoHandle& info )
{
  try
  {
    // Read the header of the file
    if ( ! this->private_->read_header() ) return false;

    // Create an information structure with the properties of this file
    info = nodereaderFileInfoHandle( new nodereaderFileInfo );
    info->set_grid_transform( this->private_->grid_transform_ );
    info->set_data_type( this->private_->pixel_type_ );
    info->set_file_type( "dicom" );
    info->set_mask_compatible( false );
  }
  catch ( ... )
  {
    this->print_error( "reader crashed when reading file." );
    return false;
  }

  return true;
}

//////////////////////////////////////////////////////////////////////

bool HDX_DCMseriesReader::get_file_data( nodereaderFileDataHandle& data )
{
  try
  {
```

```
    // Read the data from the file
    if ( !this->private_->read_data() ) return false;

    // Create a data structure with handles to the actual data in this file
    data = nodereaderFileDataHandle( new nodereaderFileData );
    data->set_data_block( this->private_->data_block_ );
    data->set_grid_transform( this->private_->grid_transform_ );
    data->set_meta_data( this->private_->meta_data_ );
    data->set_name( this->get_file_tag() );
  }
  catch ( ... )
  {
    this->print_error( "reader crashed when reading file." );
    return false;
  }

  return true;
}

////////////////////////////////////////////////////////////////////////

void HDX_DCMseriesReader::draw_slice( nodeSceneItemHandle node_item,
            const Core::Matrix& proj_mat,
            ProxyRectangleHandle rect )
{
  this->slice_shader_->set_volume_type( node_item->type() );
  this->slice_shader_->set_opacity( static_cast< float >( node_item->opacity_ ) );
  Core::VolumeSlice* volume_slice = node_item->volume_slice_.get();
  switch ( node_item->type() )
  {
  case Core::VolumeType::DATA_E:
    {
      DatanodeSceneItem* data_node_item =
        dynamic_cast<DatanodeSceneItem*>( node_item.get() );
      this->set_scale_bias( data_node_item->data_min_, data_node_item-
>data_max_,
        data_node_item->display_min_, data_node_item->display_max_ );
      this->slice_shader_->set_texture_clamp( 0.0f, 1.0f, 0.0f, 1.0f );
      this->map_slice_texture( volume_slice->get_texture(),
        static_cast<int>( volume_slice->nx() ), static_cast<int>( volume_slice->ny() ),
        volume_slice->left(), volume_slice->right(),
        volume_slice->bottom(), volume_slice->top(), proj_mat, rect );
    }
    break;
  case Core::VolumeType::MASK_E:
    {
      MasknodeSceneItem* mask_node_item =
```

```
        dynamic_cast< MasknodeSceneItem* >( node_item.get() );
      if ( rect )
      {
        this->slice_shader_->set_mask_mode( 2 );
      }
      else
      {
        // If mask fill mode is none, force the border width to be at least 1
        if ( mask_node_item->fill_ == 0 && mask_node_item->border_ == 0 )
        {
          mask_node_item->border_ = 1;
        }

        this->slice_shader_->set_mask_mode( mask_node_item->fill_ );
        this->slice_shader_->set_border_width( mask_node_item->border_ );
      }
      Core::Color color = PreferencesManager::Instance()->get_color(
mask_node_item->color_ );
      this->slice_shader_->set_mask_color( static_cast< float >( color.r() / 255 ),
        static_cast< float >( color.g() / 255 ), static_cast< float >( color.b() / 255 ) );
      this->slice_shader_->set_texture_clamp( 0.0f, 1.0f, 0.0f, 1.0f );
      this->map_slice_texture( volume_slice->get_texture(),
        static_cast<int>( volume_slice->nx() ), static_cast<int>( volume_slice->ny() ),
        volume_slice->left(), volume_slice->right(),
        volume_slice->bottom(), volume_slice->top(), proj_mat, rect );
    }
    break;
  case Core::VolumeType::LARGE_DATA_E:
    {
      LargeVolumenodeSceneItem* data_node_item =
        dynamic_cast< LargeVolumenodeSceneItem* >(node_item.get());
      this->set_scale_bias(data_node_item->data_min_, data_node_item-
>data_max_,
        data_node_item->display_min_, data_node_item->display_max_);
      const std::vector<Core::LargeVolumeBrickSliceHandle>& tiles =
data_node_item->tiles_;
      double left, right, bottom, top;
      int width, height;
      Core::Texture2DHandle texture;
      for (size_t i = 0; i < tiles.size(); ++i)
      {
        Core::LargeVolumeBrickSliceHandle tile = tiles[ i ];
        Core::BBox inner = tile->get_inner_brick_bbox();
        Core::BBox outer = tile->get_outer_brick_bbox();

        volume_slice->project_onto_slice( outer.min(), left, bottom );
        volume_slice->project_onto_slice( outer.max(), right, top );
```

```
        double ileft, iright, ibottom, itop;
        volume_slice->project_onto_slice( inner.min(), ileft, ibottom );
        volume_slice->project_onto_slice( inner.max(), iright, itop );

        texture = tile->get_texture( volume_slice->get_slice_type(),
          volume_slice->depth(), width, height, Core::ExportToString( this-
>viewer_id_ ) );
        if (texture)
        {
          this->slice_shader_->set_texture_clamp(
            Core::Max( 0.0f, static_cast<float>( ( ileft - left ) / ( right - left ) ) ),
            Core::Min( 1.0f, static_cast<float>( ( iright - left ) / ( right - left ) ) ),
            Core::Max( 0.0f, static_cast<float>( ( ibottom - bottom ) / ( top - bottom ) ) ),
            Core::Min( 1.0f, static_cast<float>( ( itop - bottom ) / ( top - bottom ) ) ) );

          this->map_large_slice_texture( texture, width, height, left, right, bottom, top,
proj_mat, rect );
        }
      }
    }
    break;
  default:
    assert( false );
    return;
  } // end switch
}

////////////////////////////////////////////////////////////////////////

void HDX_DCMseriesReader::map_large_slice_texture(Core::Texture2DHandle
slice_tex, int width, int height,
  double left, double right, double bottom, double top,
  const Core::Matrix& proj_mat, ProxyRectangleHandle rect)
{
  double slice_width = right - left;
  double slice_height = top - bottom;
  if (slice_width == 0.0 || slice_height == 0.0)
  {
    return;
  }

  Core::Texture::lock_type slice_tex_lock(slice_tex->get_mutex());
  slice_tex->bind();

  double texel_width = slice_width / width;
  double texel_height = slice_height / height;
```

```
if (rect)
{
  double tex_left = (rect->left - left) / slice_width;
  double tex_right = (rect->right - right) / slice_width + 1.0;
  double tex_bottom = (rect->bottom - bottom) / slice_height;
  double tex_top = (rect->top - top) / slice_height + 1.0;
  glBegin(GL_QUADS);
  glNormal3dv(&rect->normal[0]);
  glMultiTexCoord2d(GL_TEXTURE0, tex_left, tex_bottom);
  glVertex3dv(&rect->bottomleft[0]);
  glMultiTexCoord2d(GL_TEXTURE0, tex_right, tex_bottom);
  glVertex3dv(&rect->bottomright[0]);
  glMultiTexCoord2d(GL_TEXTURE0, tex_right, tex_top);
  glVertex3dv(&rect->topright[0]);
  glMultiTexCoord2d(GL_TEXTURE0, tex_left, tex_top);
  glVertex3dv(&rect->topleft[0]);
  glEnd();
}
else
{
  // Compute the size of the slice on screen
  Core::Vector slice_x(slice_width, 0.0, 0.0);
  slice_x = proj_mat * slice_x;
  double slice_screen_width = Core::Abs(slice_x.x()) / 2.0 * this->renderer_-
>width_;
  double slice_screen_height = slice_height / slice_width * slice_screen_width;
  float pattern_repeats_x = static_cast< float >(slice_screen_width /
PATTERN_SIZE_C);
  float pattern_repeats_y = static_cast< float >(slice_screen_height /
PATTERN_SIZE_C);
  this->slice_shader_->set_pixel_size(static_cast< float >(1.0 /
slice_screen_width),
    static_cast< float >(1.0 / slice_screen_height));
  glBegin(GL_QUADS);
  glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 0.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 0.0f, 0.0f);
  glVertex2d(left, bottom);
  glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 0.0f);
  glMultiTexCoord2f(GL_TEXTURE1, pattern_repeats_x, 0.0f);
  glVertex2d(right, bottom);
  glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 1.0f);
  glMultiTexCoord2f(GL_TEXTURE1, pattern_repeats_x, pattern_repeats_y);
  glVertex2d(right, top);
  glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 1.0f);
  glMultiTexCoord2f(GL_TEXTURE1, 0.0f, pattern_repeats_y);
  glVertex2d(left, top);
```

```
    glEnd();
   }

   slice_tex->unbind();
}

//////////////////////////////////////////////////////////////////////

void HDX_DCMseriesReader::map_slice_texture(Core::Texture2DHandle
slice_tex, int width, int height,
   double left, double right, double bottom, double top,
   const Core::Matrix& proj_mat, ProxyRectangleHandle rect)
{
   double slice_width = right - left;
   double slice_height = top - bottom;
   if (slice_width == 0.0 || slice_height == 0.0)
   {
    return;
   }

   Core::Texture::lock_type slice_tex_lock(slice_tex->get_mutex());
   slice_tex->bind();

   double texel_width = slice_width / (width - 1);
   double texel_height = slice_height / (height - 1);

   left = left - 0.5 * texel_width;
   right = right + 0.5 * texel_width;
   bottom = bottom - 0.5 * texel_height;
   top = top + 0.5 * texel_height;
   slice_width += texel_width;
   slice_height += texel_height;

   if (rect)
   {
    double tex_left = (rect->left - left) / slice_width;
    double tex_right = (rect->right - right) / slice_width + 1.0;
    double tex_bottom = (rect->bottom - bottom) / slice_height;
    double tex_top = (rect->top - top) / slice_height + 1.0;
    glBegin(GL_QUADS);
    glNormal3dv(&rect->normal[0]);
    glMultiTexCoord2d(GL_TEXTURE0, tex_left, tex_bottom);
    glVertex3dv(&rect->bottomleft[0]);
    glMultiTexCoord2d(GL_TEXTURE0, tex_right, tex_bottom);
    glVertex3dv(&rect->bottomright[0]);
    glMultiTexCoord2d(GL_TEXTURE0, tex_right, tex_top);
    glVertex3dv(&rect->topright[0]);
```

```
    glMultiTexCoord2d(GL_TEXTURE0, tex_left, tex_top);
    glVertex3dv(&rect->topleft[0]);
    glEnd();
   }
   else
   {
    // Compute the size of the slice on screen
    Core::Vector slice_x(slice_width, 0.0, 0.0);
    slice_x = proj_mat * slice_x;
    double slice_screen_width = Core::Abs(slice_x.x()) / 2.0 * this->renderer_-
>width_;
    double slice_screen_height = slice_height / slice_width * slice_screen_width;
    float pattern_repeats_x = static_cast< float >(slice_screen_width /
PATTERN_SIZE_C);
    float pattern_repeats_y = static_cast< float >(slice_screen_height /
PATTERN_SIZE_C);
    this->slice_shader_->set_pixel_size(static_cast< float >(1.0 /
slice_screen_width),
      static_cast< float >(1.0 / slice_screen_height));
    glBegin(GL_QUADS);
    glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 0.0f);
    glMultiTexCoord2f(GL_TEXTURE1, 0.0f, 0.0f);
    glVertex2d(left, bottom);
    glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 0.0f);
    glMultiTexCoord2f(GL_TEXTURE1, pattern_repeats_x, 0.0f);
    glVertex2d(right, bottom);
    glMultiTexCoord2f(GL_TEXTURE0, 1.0f, 1.0f);
    glMultiTexCoord2f(GL_TEXTURE1, pattern_repeats_x, pattern_repeats_y);
    glVertex2d(right, top);
    glMultiTexCoord2f(GL_TEXTURE0, 0.0f, 1.0f);
    glMultiTexCoord2f(GL_TEXTURE1, 0.0f, pattern_repeats_y);
    glVertex2d(left, top);
    glEnd();
   }

  slice_tex->unbind();
}

////////////////////////////////////////////////////////////////////////
//
// Node registry
//
// Registers/Deregisters HDX_DCMseriesReader node
//
////////////////////////////////////////////////////////////////////////

MStatus initializePlugin( MObject obj )
```

```
{
        MFnPlugin plugin( obj, PLUGIN_COMPANY, "1.0", "Any");
        MStatus stat = plugin.registerShape( "HDX_DCMseriesReader",
HDX_DCMseriesReader::id,

&HDX_DCMseriesReader::creator,

&HDX_DCMseriesReader::initialize,

&HDX_DCMseriesReader::creator  );
        if ( ! stat ) {
                cerr << "Failed to register node\n";
        }

        return stat;
}

MStatus uninitializePlugin( MObject obj)
{
        MFnPlugin plugin( obj );
        MStatus stat;

        stat = plugin.deregisterNode( HDX_DCMseriesReader::id );
        if ( ! stat ) {
                cerr << "Failed to deregister node : HDX_DCMseriesReader \n";
        }

        return stat;
}



////////////////////////////////////////////////////////////////////////
//
// HDX_sliceSeriesToVolume (Maya node)
//
// converts the series of sclices to a volume.
// this node allows to optionally display the generated volume in the maya
// viewport (hardware rendering)
//
////////////////////////////////////////////////////////////////////////


// Maya includes
#include <math.h>
#include <maya/MIOStream.h>
```

```
#include <maya/MFnPlugin.h>
#include <maya/MMaterial.h>
#include <maya/MSelectionList.h>
#include <maya/MSelectionMask.h>
#include <maya/MDrawData.h>
#include <maya/MMatrix.h>
#include <maya/MObjectArray.h>
#include <maya/MDagPath.h>

// Core includes
#include <Core/Math/MathFunctions.h>
#include <Core/VolumeRenderer/VolumeRendererBase.h>
#include <Core/Geometry/Algorithm.h>
#include <Core/DataBlock/StdDataBlock.h>


////////////////////////////////////////////////////////////////////////
// HDX_sliceSeriesToVolume
////////////////////////////////////////////////////////////////////////

MTypeId HDX_sliceSeriesToVolume::id( 0x8000002 );

void* HDX_sliceSeriesToVolume::creator()
{
        return new HDX_sliceSeriesToVolume();
}

MStatus HDX_sliceSeriesToVolume::initialize()
{
        return MS::kSuccess;
}

class HDX_sliceSeriesToVolume
{
        public:
                VolumeShaderSimpleHandle volume_shader_;
};

HDX_sliceSeriesToVolume::HDX_sliceSeriesToVolume() :
  private_( new HDX_sliceSeriesToVolume )
{
}

HDX_sliceSeriesToVolume::~HDX_sliceSeriesToVolume()
{
}
```

```
////////////////////////////////////////////////////////////////////////

void HDX_sliceSeriesToVolume::draw_volume( DataVolumeHandle volume, const
VolumeRenderingParam& param )
{
  std::vector< BrickEntry > brick_queue;
  this->process_volume( volume, param.sampling_rate_, param.view_,
    param.orthographic_, false, brick_queue );

  size_t num_bricks = brick_queue.size();
  if ( num_bricks == 0 )
  {
    return;
  }
  Vector voxel_size = this->get_voxel_size();

  glPushAttrib( GL_DEPTH_BUFFER_BIT | GL_POLYGON_BIT );
  glEnable( GL_DEPTH_TEST );
  glDepthMask( GL_FALSE );
  glDisable( GL_CULL_FACE );

  unsigned int old_tex_unit = Texture::GetActiveTextureUnit();
  TextureHandle diffuse_lut = param.transfer_function_->get_diffuse_lut();
  TextureHandle specular_lut = param.transfer_function_->get_specular_lut();
  Texture::lock_type diffuse_lock( diffuse_lut->get_mutex() );
  Texture::lock_type specular_lock( specular_lut->get_mutex() );
  Texture::SetActiveTextureUnit( 1 );
  diffuse_lut->bind();
  Texture::SetActiveTextureUnit( 2 );
  specular_lut->bind();
  Texture::SetActiveTextureUnit( 0 );

  this->private_->volume_shader_->enable();
  this->private_->volume_shader_->set_voxel_size(
    static_cast< float >( voxel_size[ 0 ] ),
    static_cast< float >( voxel_size[ 1 ] ),
    static_cast< float >( voxel_size[ 2 ] ) );
  this->private_->volume_shader_->set_lighting( param.enable_lighting_ );
  this->private_->volume_shader_->set_fog( param.enable_fog_ );
  this->private_->volume_shader_->set_slice_distance( static_cast< float >(
    this->get_normalized_sample_distance() ) );
  this->private_->volume_shader_->set_fog_range( static_cast< float >(
param.znear_ ),
    static_cast< float >( param.zfar_ ) );
  this->private_->volume_shader_->set_clip_plane( param.clip_plane_ );
  this->private_->volume_shader_->set_enable_clip_plane(
param.enable_clip_plane_ );
```

```
  this->private_->volume_shader_->set_enable_clipping( param.enable_clipping_
);

  glEnableClientState( GL_VERTEX_ARRAY );
  for ( size_t i = 0; i < num_bricks; ++i )
  {
    std::vector< PointF > polygon_vertices;
    std::vector< int > first_vec, count_vec;
    DataVolumeBrickHandle brick = brick_queue[ i ].brick_;
    this->slice_brick( brick, polygon_vertices, first_vec, count_vec );

    BBox texture_bbox = brick->get_texture_bbox();
    Texture3DHandle brick_texture = brick->get_texture();
    VectorF texel_size( brick->get_texel_size() );
    VectorF texture_size( texture_bbox.diagonal() );
    this->private_->volume_shader_->set_texture_bbox_min( static_cast< float >(
texture_bbox.min().x() ),
      static_cast< float >( texture_bbox.min().y() ), static_cast< float >(
texture_bbox.min().z() ) );
    this->private_->volume_shader_->set_texture_bbox_size( texture_size[ 0 ],
texture_size[ 1 ], texture_size[ 2 ] );
    this->private_->volume_shader_->set_texel_size( texel_size[ 0 ], texel_size[ 1 ],
texel_size[ 2 ] );

    Texture::lock_type tex_lock( brick_texture->get_mutex() );
    brick_texture->bind();

    glVertexPointer( 3, GL_FLOAT, 0, &polygon_vertices[ 0 ][ 0 ] );
    glMultiDrawArrays( GL_POLYGON, &first_vec[ 0 ], &count_vec[ 0 ],
      static_cast< GLsizei >( count_vec.size() ) );

    brick_texture->unbind();
  }
  glDisableClientState( GL_VERTEX_ARRAY );

  this->private_->volume_shader_->disable();

  Texture::SetActiveTextureUnit( 1 );
  diffuse_lut->unbind();
  Texture::SetActiveTextureUnit( 2 );
  specular_lut->unbind();
  Texture::SetActiveTextureUnit( old_tex_unit );
  glPopAttrib();
}
}

//////////////////////////////////////////////////////////////////////
```

```
//
// Node registry
//
// Registers/Deregisters HDX_sliceSeriesToVolume node
//
//////////////////////////////////////////////////////////////////////////

MStatus initializePlugin( MObject obj )
{
	MFnPlugin plugin( obj, PLUGIN_COMPANY, "1.0", "Any");
	MStatus stat = plugin.registerShape( "HDX_sliceSeriesToVolume",
HDX_sliceSeriesToVolume::id,

&HDX_sliceSeriesToVolume::creator,

&HDX_sliceSeriesToVolume::initialize,

&HDX_sliceSeriesToVolume::creator  );
	if ( ! stat ) {
		cerr << "Failed to register node\n";
	}

	return stat;
}

MStatus uninitializePlugin( MObject obj)
{
	MFnPlugin plugin( obj );
	MStatus stat;

	stat = plugin.deregisterNode( HDX_sliceSeriesToVolume::id );
	if ( ! stat ) {
		cerr << "Failed to deregister node : HDX_sliceSeriesToVolume \n";
	}

	return stat;
}



//////////////////////////////////////////////////////////////////////////
//
// HDX_volumeFilteringClipping (Maya node)
//
// clips and filters the volume data. mean / gaussian kernels are used for
// filtering. this node allows to optionally display the volume result in
```

```
// the maya viewport (hardware rendering)
//
////////////////////////////////////////////////////////////////////////

// Maya includes
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MFnPlugin.h>
#include <maya/MMaterial.h>
#include <maya/MSelectionList.h>
#include <maya/MSelectionMask.h>
#include <maya/MDrawData.h>
#include <maya/MMatrix.h>
#include <maya/MObjectArray.h>
#include <maya/MDagPath.h>

// Core includes
#include <Core/DataBlock/StdDataBlock.h>

////////////////////////////////////////////////////////////////////////
// HDX_volumeFilteringClipping
////////////////////////////////////////////////////////////////////////

MTypeId HDX_volumeFilteringClipping::id( 0x8000003 );

void* HDX_volumeFilteringClipping::creator()
{
        return new HDX_volumeFilteringClipping();
}

MStatus HDX_volumeFilteringClipping::initialize()
{
        return MS::kSuccess;
}

class HDX_volumeFilteringClipping
{
        public:

        void handle_kernel_changed( std::string kernel_name );

        nodeResampler* tool_;
        nodeHandle src_node_;
        nodeHandle dst_node_;
};
```

```
////////////////////////////////////////////////////////////////////////

void HDX_volumeFilteringClipping::handle_kernel_changed( std::string
kernel_name )
{
  this->tool_->has_gaussian_params_state_->set( kernel_name ==
NrrdResampleFilter::GAUSSIAN_C );
  this->tool_->has_bspline_params_state_->set( kernel_name ==
ITKResampleFilter::B_SPLINE_C );
}

////////////////////////////////////////////////////////////////////////

HDX_volumeFilteringClipping::HDX_volumeFilteringClipping( nodeHandle
src_node, nodeHandle dst_node ) :
  Core::StateHandler( "noderesampler", false ),
  private_( new nodeResamplerPrivate )
{
  this->private_->tool_ = this;
  this->private_->src_node_ = src_node;
  this->private_->dst_node_ = dst_node;

  std::vector< Core::OptionLabelPair > padding_values;
  padding_values.push_back( std::make_pair( PadValues::ZERO_C, "0" ) );
  padding_values.push_back( std::make_pair( PadValues::MIN_C, "Minimum
Value" ) );
  padding_values.push_back( std::make_pair( PadValues::MAX_C, "Maximum
Value" ) );
  this->add_state( "pad_value", this->padding_value_state_, PadValues::ZERO_C,
padding_values );

  std::vector< Core::OptionLabelPair > kernels;
  kernels.push_back( std::make_pair( NrrdResampleFilter::BOX_C, "Box" ) );
  kernels.push_back( std::make_pair( NrrdResampleFilter::TENT_C, "Tent" ) );
  kernels.push_back( std::make_pair( NrrdResampleFilter::CUBIC_CR_C, "Cubic
(Catmull-Rom)" ) );
  kernels.push_back( std::make_pair( NrrdResampleFilter::CUBIC_BS_C, "Cubic
(B-spline)" ) );
  kernels.push_back( std::make_pair( NrrdResampleFilter::QUARTIC_C, "Quartic"
) );
  kernels.push_back( std::make_pair( NrrdResampleFilter::GAUSSIAN_C,
"Gaussian" ) );
  kernels.push_back( std::make_pair( ITKResampleFilter::LINEAR_C, "Linear" ) );
  kernels.push_back( std::make_pair( ITKResampleFilter::B_SPLINE_C, "B-spline"
) );
  kernels.push_back( std::make_pair(
```

```
ITKResampleFilter::NEAREST_NEIGHBOR_C, "Nearest Neighbor" ) );
  this->add_state( "kernel", this->kernel_state_, NrrdResampleFilter::BOX_C,
kernels );

  this->add_state( "sigma", this->gauss_sigma_state_, 1.0, 1.0, 100.0, 0.01 );
  this->add_state( "cutoff", this->gauss_cutoff_state_, 1.0, 1.0, 100.0, 0.01 );
  this->add_state( "spline_order", this->spline_order_state_, 3, 0, 5, 1 );
  this->add_state( "has_gaussian_params", this->has_gaussian_params_state_,
false );
  this->add_state( "has_bspline_params", this->has_bspline_params_state_, false
);

  this->add_connection( this->kernel_state_->value_changed_signal_.connect(
    boost::bind( &nodeResamplerPrivate::handle_kernel_changed, this->private_,
_2 ) ) );
}

////////////////////////////////////////////////////////////////////////

HDX_volumeFilteringClipping::~HDX_volumeFilteringClipping()
{
  this->disconnect_all();
}

////////////////////////////////////////////////////////////////////////

void HDX_volumeFilteringClipping::run( Core::ActionContextHandle context )
{
  Core::StateEngine::lock_type lock( Core::StateEngine::GetMutex() );

  ActionResample::Dispatch( context,
                            this->private_->src_node_->get_node_id(),
                            this->private_->dst_node_->get_node_id(),
                            this->padding_value_state_->get(),
                            this->kernel_state_->get(),
                            this->gauss_sigma_state_->get(),
                            this->gauss_cutoff_state_->get(),
                            this->spline_order_state_->get(),
                            true );
}

////////////////////////////////////////////////////////////////////////
//
// Node registry
//
// Registers/Deregisters HDX_volumeFilteringClipping node
//
```

```
////////////////////////////////////////////////////////////////////////

MStatus initializePlugin( MObject obj )
{
        MFnPlugin plugin( obj, PLUGIN_COMPANY, "1.0", "Any");
        MStatus stat = plugin.registerShape( "HDX_volumeFilteringClipping",
HDX_volumeFilteringClipping::id,

&HDX_volumeFilteringClipping::creator,

&HDX_volumeFilteringClipping::initialize,

&HDX_volumeFilteringClipping::creator  );
        if ( ! stat ) {
                cerr << "Failed to register node\n";
        }

        return stat;
}

MStatus uninitializePlugin( MObject obj)
{
        MFnPlugin plugin( obj );
        MStatus stat;

        stat = plugin.deregisterNode( HDX_volumeFilteringClipping::id );
        if ( ! stat ) {
                cerr << "Failed to deregister node : HDX_volumeFilteringClipping \n";
        }

        return stat;
}



////////////////////////////////////////////////////////////////////////
//
// HDX_volumeToIsosurface (Maya node)
//
// generates the isosurface from the volume. otsu volume histogram threshold
// are used to generate the isosurface. this node allows to optionally display
// the generated isosurface in the maya viewport (hardware rendering)
//
////////////////////////////////////////////////////////////////////////
```

```
// Maya includes
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MFnPlugin.h>
#include <maya/MMaterial.h>
#include <maya/MSelectionList.h>
#include <maya/MSelectionMask.h>
#include <maya/MDrawData.h>
#include <maya/MMatrix.h>
#include <maya/MObjectArray.h>
#include <maya/MDagPath.h>

// ITK includes
#include <itkOtsuMultipleThresholdsImageFilter.h>

// Core includes
#include <Core/DataBlock/StdDataBlock.h>
#include <Core/Utils/StackVector.h>
#include <Core/Utils/Parallel.h>
#include <Core/Utils/Log.h>


//////////////////////////////////////////////////////////////////////////
// HDX_volumeToIsosurface
//////////////////////////////////////////////////////////////////////////

MTypeId HDX_volumeToIsosurface::id( 0x8000004 );

void* HDX_volumeToIsosurface::creator()
{
        return new HDX_volumeToIsosurface();
}

MStatus HDX_volumeToIsosurface::initialize()
{
        return MS::kSuccess;
}

//////////////////////////////////////////////////////////////////////////

OtsuThresholdFilter::OtsuThresholdFilter( const std::string& toolid ) :
  SingleTargetTool( Core::VolumeType::DATA_E, toolid )
{
  // Need to set ranges and default values for all parameters
  add_state( "amount", this->amount_state_, 1, 1, 4, 1 );
}
```

```
OtsuThresholdFilter::~OtsuThresholdFilter()
{
  disconnect_all();
}

void OtsuThresholdFilter::execute( Core::ActionContextHandle context )
{
  Core::StateEngine::lock_type lock( Core::StateEngine::GetMutex() );

  ActionOtsuThresholdFilter::Dispatch( context,
    this->target_node_state_->get(),
    this->amount_state_->get() );
}

////////////////////////////////////////////////////////////////////////

// Marching Cubes tutorial:
http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/

typedef struct {
  int edges_[15]; // Vertex indices for at most 5 triangles
  int num_triangles_; // Last number in each table entry
} MarchingCubesTableType;

// Precalculated array of 256 possible polygon configurations (2^8 = 256) within
the cube
// 256x16 table of integer values, which are used as indices for the array of 12
points of
// intersection. It defines the right order to connect the intersected edges to form
triangles.
// The process for one cell stops when index of -1 is returned from the table,
forming a maximum of
// 5 triangles.

const MarchingCubesTableType MARCHING_CUBES_TABLE_C[] = {
  {{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 0},
  {{ 0,  3,  8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 0,  9,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 1,  3,  8,  9,  1,  8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 1, 11,  2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 0,  3,  8,  1, 11,  2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 9, 11,  2,  0,  9,  2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 2,  3,  8,  2,  8, 11, 11,  8,  9, -1, -1, -1, -1, -1, -1, -1}, 3},
  {{ 3,  2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 0,  2, 10,  8,  0, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 1,  0,  9,  2, 10,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 1,  2, 10,  1, 10,  9,  9, 10,  8, -1, -1, -1, -1, -1, -1, -1}, 3},
```

{{ 3, 1, 11, 10, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 1, 11, 0, 11, 8, 8, 11, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 0, 9, 3, 9, 10, 10, 9, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 11, 8, 11, 10, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 8, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 4, 0, 3, 7, 4, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 9, 1, 8, 7, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 9, 1, 4, 1, 7, 7, 1, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 11, 2, 8, 7, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 3, 7, 4, 3, 4, 0, 1, 11, 2, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 11, 2, 9, 2, 0, 8, 7, 4, -1, -1, -1, -1, -1, -1}, 3},
{{ 2, 9, 11, 2, 7, 9, 2, 3, 7, 7, 4, 9, -1, -1, -1}, 4},
{{ 8, 7, 4, 3, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{10, 7, 4, 10, 4, 2, 2, 4, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 1, 0, 8, 7, 4, 2, 10, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 4, 10, 7, 9, 10, 4, 9, 2, 10, 9, 1, 2, -1, -1, -1}, 4},
{{ 3, 1, 11, 3, 11, 10, 7, 4, 8, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 11, 10, 1, 10, 4, 1, 4, 0, 7, 4, 10, -1, -1, -1}, 4},
{{ 4, 8, 7, 9, 10, 0, 9, 11, 10, 10, 3, 0, -1, -1, -1}, 4},
{{ 4, 10, 7, 4, 9, 10, 9, 11, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 9, 4, 5, 0, 3, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 4, 5, 1, 0, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 8, 4, 5, 8, 5, 3, 3, 5, 1, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 11, 2, 9, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 3, 8, 0, 1, 11, 2, 4, 5, 9, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 11, 2, 5, 2, 4, 4, 2, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 2, 5, 11, 3, 5, 2, 3, 4, 5, 3, 8, 4, -1, -1, -1}, 4},
{{ 9, 4, 5, 2, 10, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 2, 10, 0, 10, 8, 4, 5, 9, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 4, 5, 0, 5, 1, 2, 10, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 2, 5, 1, 2, 8, 5, 2, 10, 8, 4, 5, 8, -1, -1, -1}, 4},
{{11, 10, 3, 11, 3, 1, 9, 4, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 4, 5, 9, 0, 1, 8, 8, 1, 11, 8, 11, 10, -1, -1, -1}, 4},
{{ 5, 0, 4, 5, 10, 0, 5, 11, 10, 10, 3, 0, -1, -1, -1}, 4},
{{ 5, 8, 4, 5, 11, 8, 11, 10, 8, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 8, 7, 5, 9, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 9, 0, 3, 9, 3, 5, 5, 3, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 8, 7, 0, 7, 1, 1, 7, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 9, 8, 7, 9, 7, 5, 11, 2, 1, -1, -1, -1, -1, -1, -1}, 3},
{{11, 2, 1, 9, 0, 5, 5, 0, 3, 5, 3, 7, -1, -1, -1}, 4},
{{ 8, 2, 0, 8, 5, 2, 8, 7, 5, 11, 2, 5, -1, -1, -1}, 4},
{{ 2, 5, 11, 2, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 7, 5, 9, 7, 9, 8, 3, 2, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 7, 5, 9, 2, 7, 9, 0, 2, 2, 10, 7, -1, -1, -1}, 4},
{{ 2, 10, 3, 0, 8, 1, 1, 8, 7, 1, 7, 5, -1, -1, -1}, 4},

{{10, 1, 2, 10, 7, 1, 7, 5, 1, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 8, 5, 8, 7, 5, 11, 3, 1, 11, 10, 3, -1, -1, -1}, 4},
{{ 5, 0, 7, 5, 9, 0, 7, 0, 10, 1, 11, 0, 10, 0, 11}, 5},
{{10, 0, 11, 10, 3, 0, 11, 0, 5, 8, 7, 0, 5, 0, 7}, 5},
{{10, 5, 11, 7, 5, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{11, 5, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 0, 3, 8, 5, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 9, 1, 0, 5, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 3, 8, 1, 8, 9, 5, 6, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 5, 6, 2, 1, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 5, 6, 1, 6, 2, 3, 8, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 5, 6, 9, 6, 0, 0, 6, 2, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 8, 9, 5, 2, 8, 5, 6, 2, 3, 8, 2, -1, -1, -1}, 4},
{{ 2, 10, 3, 11, 5, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{10, 8, 0, 10, 0, 2, 11, 5, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 9, 1, 2, 10, 3, 5, 6, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 6, 11, 1, 2, 9, 9, 2, 10, 9, 10, 8, -1, -1, -1}, 4},
{{ 6, 10, 3, 6, 3, 5, 5, 3, 1, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 10, 8, 0, 5, 10, 0, 1, 5, 5, 6, 10, -1, -1, -1}, 4},
{{ 3, 6, 10, 0, 6, 3, 0, 5, 6, 0, 9, 5, -1, -1, -1}, 4},
{{ 6, 9, 5, 6, 10, 9, 10, 8, 9, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 6, 11, 4, 8, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 0, 3, 4, 3, 7, 6, 11, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 0, 9, 5, 6, 11, 8, 7, 4, -1, -1, -1, -1, -1, -1}, 3},
{{11, 5, 6, 1, 7, 9, 1, 3, 7, 7, 4, 9, -1, -1, -1}, 4},
{{ 6, 2, 1, 6, 1, 5, 4, 8, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 5, 2, 5, 6, 2, 3, 4, 0, 3, 7, 4, -1, -1, -1}, 4},
{{ 8, 7, 4, 9, 5, 0, 0, 5, 6, 0, 6, 2, -1, -1, -1}, 4},
{{ 7, 9, 3, 7, 4, 9, 3, 9, 2, 5, 6, 9, 2, 9, 6}, 5},
{{ 3, 2, 10, 7, 4, 8, 11, 5, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 6, 11, 4, 2, 7, 4, 0, 2, 2, 10, 7, -1, -1, -1}, 4},
{{ 0, 9, 1, 4, 8, 7, 2, 10, 3, 5, 6, 11, -1, -1, -1}, 4},
{{ 9, 1, 2, 9, 2, 10, 9, 10, 4, 7, 4, 10, 5, 6, 11}, 5},
{{ 8, 7, 4, 3, 5, 10, 3, 1, 5, 5, 6, 10, -1, -1, -1}, 4},
{{ 5, 10, 1, 5, 6, 10, 1, 10, 0, 7, 4, 10, 0, 10, 4}, 5},
{{ 0, 9, 5, 0, 5, 6, 0, 6, 3, 10, 3, 6, 8, 7, 4}, 5},
{{ 6, 9, 5, 6, 10, 9, 4, 9, 7, 7, 9, 10, -1, -1, -1}, 4},
{{11, 9, 4, 6, 11, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 6, 11, 4, 11, 9, 0, 3, 8, -1, -1, -1, -1, -1, -1}, 3},
{{11, 1, 0, 11, 0, 6, 6, 0, 4, -1, -1, -1, -1, -1, -1}, 3},
{{ 8, 1, 3, 8, 6, 1, 8, 4, 6, 6, 11, 1, -1, -1, -1}, 4},
{{ 1, 9, 4, 1, 4, 2, 2, 4, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 8, 0, 1, 9, 2, 2, 9, 4, 2, 4, 6, -1, -1, -1}, 4},
{{ 0, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 8, 2, 3, 8, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1}, 3},
{{11, 9, 4, 11, 4, 6, 10, 3, 2, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 2, 8, 2, 10, 8, 4, 11, 9, 4, 6, 11, -1, -1, -1}, 4},

{{ 3, 2, 10, 0, 6, 1, 0, 4, 6, 6, 11, 1, -1, -1, -1}, 4},
{{ 6, 1, 4, 6, 11, 1, 4, 1, 8, 2, 10, 1, 8, 1, 10}, 5},
{{ 9, 4, 6, 9, 6, 3, 9, 3, 1, 10, 3, 6, -1, -1, -1}, 4},
{{ 8, 1, 10, 8, 0, 1, 10, 1, 6, 9, 4, 1, 6, 1, 4}, 5},
{{ 3, 6, 10, 3, 0, 6, 0, 4, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 6, 8, 4, 10, 8, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 7, 6, 11, 7, 11, 8, 8, 11, 9, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 3, 7, 0, 7, 11, 0, 11, 9, 6, 11, 7, -1, -1, -1}, 4},
{{11, 7, 6, 1, 7, 11, 1, 8, 7, 1, 0, 8, -1, -1, -1}, 4},
{{11, 7, 6, 11, 1, 7, 1, 3, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 6, 2, 1, 8, 6, 1, 9, 8, 8, 7, 6, -1, -1, -1}, 4},
{{ 2, 9, 6, 2, 1, 9, 6, 9, 7, 0, 3, 9, 7, 9, 3}, 5},
{{ 7, 0, 8, 7, 6, 0, 6, 2, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 7, 2, 3, 6, 2, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 2, 10, 3, 11, 8, 6, 11, 9, 8, 8, 7, 6, -1, -1, -1}, 4},
{{ 2, 7, 0, 2, 10, 7, 0, 7, 9, 6, 11, 7, 9, 7, 11}, 5},
{{ 1, 0, 8, 1, 8, 7, 1, 7, 11, 6, 11, 7, 2, 10, 3}, 5},
{{10, 1, 2, 10, 7, 1, 11, 1, 6, 6, 1, 7, -1, -1, -1}, 4},
{{ 8, 6, 9, 8, 7, 6, 9, 6, 1, 10, 3, 6, 1, 6, 3}, 5},
{{ 0, 1, 9, 10, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 7, 0, 8, 7, 6, 0, 3, 0, 10, 10, 0, 6, -1, -1, -1}, 4},
{{ 7, 6, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 7, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 3, 8, 0, 10, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 9, 1, 10, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 8, 9, 1, 8, 1, 3, 10, 6, 7, -1, -1, -1, -1, -1, -1}, 3},
{{11, 2, 1, 6, 7, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 11, 2, 3, 8, 0, 6, 7, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 2, 0, 9, 2, 9, 11, 6, 7, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 6, 7, 10, 2, 3, 11, 11, 3, 8, 11, 8, 9, -1, -1, -1}, 4},
{{ 7, 3, 2, 6, 7, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 7, 8, 0, 7, 0, 6, 6, 0, 2, -1, -1, -1, -1, -1, -1}, 3},
{{ 2, 6, 7, 2, 7, 3, 0, 9, 1, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 2, 6, 1, 6, 8, 1, 8, 9, 8, 6, 7, -1, -1, -1}, 4},
{{11, 6, 7, 11, 7, 1, 1, 7, 3, -1, -1, -1, -1, -1, -1}, 3},
{{11, 6, 7, 1, 11, 7, 1, 7, 8, 1, 8, 0, -1, -1, -1}, 4},
{{ 0, 7, 3, 0, 11, 7, 0, 9, 11, 6, 7, 11, -1, -1, -1}, 4},
{{ 7, 11, 6, 7, 8, 11, 8, 9, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 6, 4, 8, 10, 6, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 3, 10, 6, 3, 6, 0, 0, 6, 4, -1, -1, -1, -1, -1, -1}, 3},
{{ 8, 10, 6, 8, 6, 4, 9, 1, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 6, 4, 9, 3, 6, 9, 1, 3, 10, 6, 3, -1, -1, -1}, 4},
{{ 6, 4, 8, 6, 8, 10, 2, 1, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 11, 2, 3, 10, 0, 0, 10, 6, 0, 6, 4, -1, -1, -1}, 4},
{{ 4, 8, 10, 4, 10, 6, 0, 9, 2, 2, 9, 11, -1, -1, -1}, 4},
{{11, 3, 9, 11, 2, 3, 9, 3, 4, 10, 6, 3, 4, 3, 6}, 5},
{{ 8, 3, 2, 8, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1}, 3},

{{ 0, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 0, 9, 2, 4, 3, 2, 6, 4, 4, 8, 3, -1, -1, -1}, 4},
{{ 1, 4, 9, 1, 2, 4, 2, 6, 4, -1, -1, -1, -1, -1, -1}, 3},
{{ 8, 3, 1, 8, 1, 6, 8, 6, 4, 6, 1, 11, -1, -1, -1}, 4},
{{11, 0, 1, 11, 6, 0, 6, 4, 0, -1, -1, -1, -1, -1, -1}, 3},
{{ 4, 3, 6, 4, 8, 3, 6, 3, 11, 0, 9, 3, 11, 3, 9}, 5},
{{11, 4, 9, 6, 4, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 5, 9, 7, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 3, 8, 4, 5, 9, 10, 6, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 1, 0, 5, 0, 4, 7, 10, 6, -1, -1, -1, -1, -1, -1}, 3},
{{10, 6, 7, 8, 4, 3, 3, 4, 5, 3, 5, 1, -1, -1, -1}, 4},
{{ 9, 4, 5, 11, 2, 1, 7, 10, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 6, 7, 10, 1, 11, 2, 0, 3, 8, 4, 5, 9, -1, -1, -1}, 4},
{{ 7, 10, 6, 5, 11, 4, 4, 11, 2, 4, 2, 0, -1, -1, -1}, 4},
{{ 3, 8, 4, 3, 4, 5, 3, 5, 2, 11, 2, 5, 10, 6, 7}, 5},
{{ 7, 3, 2, 7, 2, 6, 5, 9, 4, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 4, 5, 0, 6, 8, 0, 2, 6, 6, 7, 8, -1, -1, -1}, 4},
{{ 3, 2, 6, 3, 6, 7, 1, 0, 5, 5, 0, 4, -1, -1, -1}, 4},
{{ 6, 8, 2, 6, 7, 8, 2, 8, 1, 4, 5, 8, 1, 8, 5}, 5},
{{ 9, 4, 5, 11, 6, 1, 1, 6, 7, 1, 7, 3, -1, -1, -1}, 4},
{{ 1, 11, 6, 1, 6, 7, 1, 7, 0, 8, 0, 7, 9, 4, 5}, 5},
{{ 4, 11, 0, 4, 5, 11, 0, 11, 3, 6, 7, 11, 3, 11, 7}, 5},
{{ 7, 11, 6, 7, 8, 11, 5, 11, 4, 4, 11, 8, -1, -1, -1}, 4},
{{ 6, 5, 9, 6, 9, 10, 10, 9, 8, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 10, 6, 0, 3, 6, 0, 6, 5, 0, 5, 9, -1, -1, -1}, 4},
{{ 0, 8, 10, 0, 10, 5, 0, 5, 1, 5, 10, 6, -1, -1, -1}, 4},
{{ 6, 3, 10, 6, 5, 3, 5, 1, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 11, 2, 9, 10, 5, 9, 8, 10, 10, 6, 5, -1, -1, -1}, 4},
{{ 0, 3, 10, 0, 10, 6, 0, 6, 9, 5, 9, 6, 1, 11, 2}, 5},
{{10, 5, 8, 10, 6, 5, 8, 5, 0, 11, 2, 5, 0, 5, 2}, 5},
{{ 6, 3, 10, 6, 5, 3, 2, 3, 11, 11, 3, 5, -1, -1, -1}, 4},
{{ 5, 9, 8, 5, 8, 2, 5, 2, 6, 3, 2, 8, -1, -1, -1}, 4},
{{ 9, 6, 5, 9, 0, 6, 0, 2, 6, -1, -1, -1, -1, -1, -1}, 3},
{{ 1, 8, 5, 1, 0, 8, 5, 8, 6, 3, 2, 8, 6, 8, 2}, 5},
{{ 1, 6, 5, 2, 6, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 6, 3, 1, 11, 6, 3, 6, 8, 5, 9, 6, 8, 6, 9}, 5},
{{11, 0, 1, 11, 6, 0, 9, 0, 5, 5, 0, 6, -1, -1, -1}, 4},
{{ 0, 8, 3, 5, 11, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{11, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{10, 11, 5, 7, 10, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{10, 11, 5, 10, 5, 7, 8, 0, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 7, 10, 5, 10, 11, 1, 0, 9, -1, -1, -1, -1, -1, -1}, 3},
{{11, 5, 7, 11, 7, 10, 9, 1, 8, 8, 1, 3, -1, -1, -1}, 4},
{{10, 2, 1, 10, 1, 7, 7, 1, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 3, 8, 1, 7, 2, 1, 5, 7, 7, 10, 2, -1, -1, -1}, 4},
{{ 9, 5, 7, 9, 7, 2, 9, 2, 0, 2, 7, 10, -1, -1, -1}, 4},
{{ 7, 2, 5, 7, 10, 2, 5, 2, 9, 3, 8, 2, 9, 2, 8}, 5},

{{ 2, 11, 5, 2, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 8, 0, 2, 8, 2, 5, 8, 5, 7, 11, 5, 2, -1, -1, -1}, 4},
{{ 9, 1, 0, 5, 3, 11, 5, 7, 3, 3, 2, 11, -1, -1, -1}, 4},
{{ 9, 2, 8, 9, 1, 2, 8, 2, 7, 11, 5, 2, 7, 2, 5}, 5},
{{ 1, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 0, 7, 8, 0, 1, 7, 1, 5, 7, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 3, 0, 9, 5, 3, 5, 7, 3, -1, -1, -1, -1, -1, -1}, 3},
{{ 9, 7, 8, 5, 7, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 5, 4, 8, 5, 8, 11, 11, 8, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 5, 4, 0, 5, 0, 10, 5, 10, 11, 10, 0, 3, -1, -1, -1}, 4},
{{ 0, 9, 1, 8, 11, 4, 8, 10, 11, 11, 5, 4, -1, -1, -1}, 4},
{{11, 4, 10, 11, 5, 4, 10, 4, 3, 9, 1, 4, 3, 4, 1}, 5},
{{ 2, 1, 5, 2, 5, 8, 2, 8, 10, 4, 8, 5, -1, -1, -1}, 4},
{{ 0, 10, 4, 0, 3, 10, 4, 10, 5, 2, 1, 10, 5, 10, 1}, 5},
{{ 0, 5, 2, 0, 9, 5, 2, 5, 10, 4, 8, 5, 10, 5, 8}, 5},
{{ 9, 5, 4, 2, 3, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 2, 11, 5, 3, 2, 5, 3, 5, 4, 3, 4, 8, -1, -1, -1}, 4},
{{ 5, 2, 11, 5, 4, 2, 4, 0, 2, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 2, 11, 3, 11, 5, 3, 5, 8, 4, 8, 5, 0, 9, 1}, 5},
{{ 5, 2, 11, 5, 4, 2, 1, 2, 9, 9, 2, 4, -1, -1, -1}, 4},
{{ 8, 5, 4, 8, 3, 5, 3, 1, 5, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 5, 4, 1, 5, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 8, 5, 4, 8, 3, 5, 9, 5, 0, 0, 5, 3, -1, -1, -1}, 4},
{{ 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 4, 7, 10, 4, 10, 9, 9, 10, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 3, 8, 4, 7, 9, 9, 7, 10, 9, 10, 11, -1, -1, -1}, 4},
{{ 1, 10, 11, 1, 4, 10, 1, 0, 4, 7, 10, 4, -1, -1, -1}, 4},
{{ 3, 4, 1, 3, 8, 4, 1, 4, 11, 7, 10, 4, 11, 4, 10}, 5},
{{ 4, 7, 10, 9, 4, 10, 9, 10, 2, 9, 2, 1, -1, -1, -1}, 4},
{{ 9, 4, 7, 9, 7, 10, 9, 10, 1, 2, 1, 10, 0, 3, 8}, 5},
{{10, 4, 7, 10, 2, 4, 2, 0, 4, -1, -1, -1, -1, -1, -1}, 3},
{{10, 4, 7, 10, 2, 4, 8, 4, 3, 3, 4, 2, -1, -1, -1}, 4},
{{ 2, 11, 9, 2, 9, 7, 2, 7, 3, 7, 9, 4, -1, -1, -1}, 4},
{{ 9, 7, 11, 9, 4, 7, 11, 7, 2, 8, 0, 7, 2, 7, 0}, 5},
{{ 3, 11, 7, 3, 2, 11, 7, 11, 4, 1, 0, 11, 4, 11, 0}, 5},
{{ 1, 2, 11, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 1, 9, 4, 7, 1, 7, 3, 1, -1, -1, -1, -1, -1, -1}, 3},
{{ 4, 1, 9, 4, 7, 1, 0, 1, 8, 8, 1, 7, -1, -1, -1}, 4},
{{ 4, 3, 0, 7, 3, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
{{ 9, 8, 11, 11, 8, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 3, 9, 0, 3, 10, 9, 10, 11, 9, -1, -1, -1, -1, -1, -1}, 3},
{{ 0, 11, 1, 0, 8, 11, 8, 10, 11, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 11, 1, 10, 11, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
{{ 1, 10, 2, 1, 9, 10, 9, 8, 10, -1, -1, -1, -1, -1, -1}, 3},
{{ 3, 9, 0, 3, 10, 9, 1, 9, 2, 2, 9, 10, -1, -1, -1}, 4},
{{ 0, 10, 2, 8, 10, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},

```
  {{ 3, 10,  2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 2,  8,  3,  2, 11,  8, 11,  9,  8, -1, -1, -1, -1, -1, -1}, 3},
  {{ 9,  2, 11,  0,  2,  9, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 2,  8,  3,  2, 11,  8,  0,  8,  1,  1,  8, 11, -1, -1, -1}, 4},
  {{ 1,  2, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 1,  8,  3,  9,  8,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 2},
  {{ 0,  1,  9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{ 0,  8,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},
  {{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 0}};

typedef struct {
  int points_[12]; // Vertex indices for at most 4 triangles
  int num_triangles_; // Last number in each table entry
} CappingTableType;

// Precalculated array of 16 possible polygon configurations (2^4 = 16) within the
cell.
// 16x13 table of integer values, which are used as indices for the array of 8 points
(4 vertices, 4 edges) of
// intersection. It defines the right order to connect the intersected edges to form
triangles.
// The process for one cell stops when index of -1 is returned from the table,
forming a maximum of
// 4 triangles.
// For example: {{tri1.p1, tri1.p2, tri1.p3, tri2.p1, tri2.p2, tri2.p3, tri3.p1, tri3.p2,
tri3.p3,
// tri4.p1, tri4,p2, tri4.p3}, num_tri}

const CappingTableType CAPPING_TABLE_C[] = {
  {{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 0},  // 0
  {{0, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},   // 1
  {{4, 1, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},   // 2
  {{0, 1, 7, 7, 1, 5, -1, -1, -1, -1, -1, -1, -1}, 2},    // 3
  {{6, 5, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},   // 4
  {{0, 4, 7, 7, 4, 5, 7, 5, 6, 6, 5, 2}, 4},        // 5
  {{4, 1, 6, 6, 1, 2, -1, -1, -1, -1, -1, -1, -1}, 2},    // 6
  {{0, 6, 7, 0, 1, 6, 6, 1, 2, -1, -1, -1, -1}, 3},     // 7
  {{7, 6, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1}, 1},   // 8
  {{0, 4, 3, 3, 4, 6, -1, -1, -1, -1, -1, -1, -1}, 2},    // 9
  {{7, 6, 3, 7, 4, 6, 4, 5, 6, 4, 1, 5}, 4},        // 10
  {{0, 6, 3, 0, 5, 6, 0, 1, 5, -1, -1, -1, -1}, 3},     // 11
  {{7, 5, 3, 3, 5, 2, -1, -1, -1, -1, -1, -1, -1}, 2},    // 12
  {{0, 5, 3, 0, 4, 5, 3, 5, 2, -1, -1, -1, -1}, 3},     // 13
  {{7, 4, 3, 3, 4, 2, 4, 1, 2, -1, -1, -1, -1}, 3},     // 14
  {{0, 1, 3, 3, 1, 2, -1, -1, -1, -1, -1, -1, -1}, 2}};   // 15

class VertexBufferBatch
```

```
{
public:
  VertexAttribArrayBufferHandle vertex_buffer_;
  VertexAttribArrayBufferHandle normal_buffer_;
  VertexAttribArrayBufferHandle value_buffer_;
  ElementArrayBufferHandle faces_buffer_;
};

typedef boost::shared_ptr< VertexBufferBatch > VertexBufferBatchHandle;

class HDX_volumeToIsosurface
{

public:
  void downsample_setup( int num_threads, double quality_factor );

  // PARALLEL_DOWNSAMPLE:
  // Downsample mask prior to computing the isosurface in order to reduce the
mesh to speed up
  // rendering.
  void parallel_downsample_mask( int thread, int num_threads, boost::barrier&
barrier,
    double quality_factor );

  // Copy values to members just to simplify and shorten code.  Must be called
after downsample
  // and before face computation.
  void compute_setup();

  // SETUP:
  // Setup the algorithm and the buffers for face computation
  void compute_faces_setup( int num_threads );

  // PARALLEL_COMPUTE_FACES:
  // Parallelized isosurface computation algorithm
  void parallel_compute_faces( int thread, int num_threads, boost::barrier& barrier
);

  void translate_cap_coords( int cap_num, float i, float j, float& x, float& y, float& z
);

  size_t get_data_index( float x, float y, float z );

  // COMPUTE_CAP_FACES:
  // Compute the "cap" faces at the boundary of the mask volume to handle the
case where the
  // mask goes all the way to the boundary.  Otherwise, we end up with holes in the
```

```
  // isosurface at the boundary.  The cap faces are computed as separate
geometry so that they can
  // be turned on/off independently from the rest of the isosurface.
  void compute_cap_faces();

  // PARALLEL_COMPUTE_NORMALS:
  // Parallelized isosurface normal computation algorithm
  void parallel_compute_normals( int thread, int num_threads,  boost::barrier&
barrier );

  // UPLOAD_TO_VERTEX_BUFFER:
  void upload_to_vertex_buffer();

  void reset();


  // Pointer to public Isosurface -- needed to give access to public signals
  Isosurface* isosurface_;

  // Downsample params
  MaskVolumeHandle downsample_mask_volume_;
  int neighborhood_size_;
  size_t zsize_;
  size_t total_neighborhoods_;

  // Input to isosurface computation, not downsampled
  MaskVolumeHandle orig_mask_volume_;
  // Mask volume to be used for isosurface computation.  May point to original
volume or
  // downsampled volume.
  MaskVolumeHandle compute_mask_volume_;
  unsigned char mask_value_; // Same for original volume and downsampled
volume

  // Output mesh
  PointFVector points_;
  VectorFVector normals_;
  UIntVector faces_; // unsigned int because GL expects this
  FloatVector values_; // Should be in range [0, 1]
  float area_; // Surface area of the isosurface

  // Single colormap shared by all isosurfaces
  ColorMapHandle color_map_;

  // Algorithm data & buffers
  unsigned char* data_; // Mask data is stored in bit-plane (8 masks per data block)
  size_t nx_, ny_, nz_; // Mask dimensions of original or downsampled volume
```

```
depending on quality
  size_t elem_nx_, elem_ny_, elem_nz_; // Number of (marching) cubes

  UCharVector type_buffer_;
  std::vector< UIntVector > edge_buffer_;

  UIntVector min_point_index_;
  UIntVector max_point_index_;
  UIntVector min_face_index_;
  UIntVector max_face_index_;

  std::vector< std::pair< unsigned int, unsigned int > > part_points_;
  std::vector< std::pair< unsigned int, unsigned int > > part_faces_;
  std::vector< UIntVector > part_indices_;

  std::vector< PointFVector > new_points_;
  std::vector< std::vector< StackVector< size_t, 3 > > > new_elems_;
  FloatVector new_elem_areas_;

  IVector front_offset_;
  IVector back_offset_;
  size_t global_point_cnt_;

  unsigned int prev_point_min_;
  unsigned int prev_point_max_;

  std::vector< VertexBufferBatchHandle > vbo_batches_;
  bool vbo_available_;
  bool surface_changed_;
  bool values_changed_;

  bool need_abort_;
  boost::function< bool () > check_abort_;

  const static double COMPUTE_PERCENT_PROGRESS_C;
  const static double NORMAL_PERCENT_PROGRESS_C;
  const static double PARTITION_PERCENT_PROGRESS_C;
};

// Initialize static variables
const double HDX_volumeToIsosurface::COMPUTE_PERCENT_PROGRESS_C
= 0.8;
const double HDX_volumeToIsosurface::NORMAL_PERCENT_PROGRESS_C =
0.05;
const double HDX_volumeToIsosurface::PARTITION_PERCENT_PROGRESS_C
= 0.15;
```

```
////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::downsample_setup( int num_threads, double
quality_factor )
{
  this->nx_ = this->orig_mask_volume_->get_mask_data_block()->get_nx();
  this->ny_ = this->orig_mask_volume_->get_mask_data_block()->get_ny();
  this->nz_ = this->orig_mask_volume_->get_mask_data_block()->get_nz();

  // Mask data is stored in bit-plane (8 masks per data block)
  this->data_ = this->orig_mask_volume_->get_mask_data_block()-
>get_mask_data();

  // Bit where mask bit is stored
  this->mask_value_ = this->orig_mask_volume_->get_mask_data_block()-
>get_mask_value();

  // Create downsampled mask to store results
  this->neighborhood_size_ = static_cast< int >( 1.0 / quality_factor );
  size_t downsampled_nx = this->nx_ / this->neighborhood_size_;
  size_t downsampled_ny = this->ny_ / this->neighborhood_size_;
  size_t downsampled_nz = this->nz_ / this->neighborhood_size_;

  // Normally MaskDataBlocks should be registered with the
MathDataBlockManager, but in this
  // case we are only using this as a temporary object and do not want to share the
mask with
  // other masks since we would then have to carefully lock/unlock it during use.
  MaskDataBlockHandle mask_data_block( new MaskDataBlock(
    StdDataBlock::New( downsampled_nx, downsampled_ny, downsampled_nz,
DataType::UCHAR_E ),
    this->orig_mask_volume_->get_mask_data_block()->get_mask_bit() ) );

  // Downsampled mask needs to be scaled up to fill the same geometric space as
the original
  // mask.
  GridTransform grid_transform = this->orig_mask_volume_-
>get_grid_transform();
  double transform_scale = static_cast< double >( this->neighborhood_size_ );
  grid_transform.post_scale( Vector( transform_scale, transform_scale,
transform_scale ) );

  this->downsample_mask_volume_ = MaskVolumeHandle( new MaskVolume(
grid_transform,
    mask_data_block ) );

  // Point to downsampled mask rather than original mask
```

```
  this->compute_mask_volume_ = this->downsample_mask_volume_;

  // For parallelization, divide mask into slabs along z.  No synchronization is
needed.
  this->zsize_ = static_cast< size_t >( this->nz_ / num_threads );
  // Make sure this zsize will cover all the data
  if ( this->zsize_ * num_threads < this->nz_ )
  {
    this->zsize_++;
  }
  int remainder = this->zsize_ % this->neighborhood_size_;
  // Round up to the next neighborhood increment -- leaves least work for last
thread
  if( remainder != 0 )
  {
    this->zsize_ += ( this->neighborhood_size_ - remainder );
  }

  size_t x_neighborhoods = this->nx_ / this->neighborhood_size_;
  size_t y_neighborhoods = this->ny_ / this->neighborhood_size_;
  size_t z_neighborhoods = this->zsize_ / this->neighborhood_size_;
  this->total_neighborhoods_ = x_neighborhoods * y_neighborhoods *
z_neighborhoods;
}

////////////////////////////////////////////////////////////////////////

/*
Allow downsampling by only half, quarter, and eighth.  When downsampling by
half, a 2x2x2
neighborhood of nodes is downsampled to a single node.  If at least one
neighborhood node is "on",
result is "on".  This method was chosen to prevent holes in the downsampled data.
*/
void HDX_volumeToIsosurface::parallel_downsample_mask( int thread, int
num_threads,
  boost::barrier& barrier, double quality_factor )
{
  // Only need to setup once
  if( thread == 0 )
  {
    this->downsample_setup( num_threads, quality_factor );
  }

  // All threads must wait for setup to complete
  barrier.wait();
```

```
  // Different thread process slabs along the z axis.  Each slab contains one or
more
  // neighborhoods to be downsampled.
  // [nzstart, nzend)
  size_t nzstart = thread * this->zsize_;
  size_t nzend = ( thread + 1 ) * this->zsize_;
  if ( nzend > this->nz_ )
  {
    nzend = this->nz_;
  }

  unsigned char* downsampled_data =
    this->downsample_mask_volume_->get_mask_data_block()->get_mask_data();

  size_t z_offset = this->nx_ * this->ny_;
  unsigned char not_mask_value = ~( this->mask_value_ );

  size_t target_index = thread * this->total_neighborhoods_;
  size_t x_start_end = this->nx_ - this->neighborhood_size_ + 1;
  size_t y_start_end = this->ny_ - this->neighborhood_size_ + 1;
  size_t z_start_end = nzend - this->neighborhood_size_ + 1;

  // Loop over neighborhoods, chop off border values
  for ( size_t z_start = nzstart; z_start < z_start_end; z_start += this-
>neighborhood_size_ )
  {
    for ( size_t y_start = 0; y_start < y_start_end; y_start += this-
>neighborhood_size_ )
    {
      for ( size_t x_start = 0; x_start < x_start_end; x_start += this-
>neighborhood_size_ )
      {
        // Clear entry initially
        downsampled_data[ target_index ] &= not_mask_value;

        bool stop = false;
        size_t x_end = x_start + this->neighborhood_size_;
        size_t y_end = y_start + this->neighborhood_size_;
        size_t z_end = z_start + this->neighborhood_size_;

        // Loop over neighbors based on corner index
        for( size_t z = z_start; z < z_end && !stop; z++ )
        {
          for( size_t y = y_start; y < y_end && !stop; y++ )
          {
            for( size_t x = x_start; x < x_end && !stop; x++ )
            {
```

```
            size_t index = ( ( z_offset ) * z ) + ( this->nx_ * y ) + x;

            // If at least one neighborhood node is "on", result is "on"
            if ( this->data_[ index ] & this->mask_value_ ) // Node "on"
            {
              // Turn on mask value
              downsampled_data[ target_index ] |= this->mask_value_;
              // Short-circuit
              stop = true;
            }
          }
        }
      }
      target_index++;
    }
  }
 }
}

////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::compute_setup()
{
  this->nx_ = this->compute_mask_volume_->get_mask_data_block()->get_nx();
  this->ny_ = this->compute_mask_volume_->get_mask_data_block()->get_ny();
  this->nz_ = this->compute_mask_volume_->get_mask_data_block()->get_nz();

  // Mask data is stored in bit-plane (8 masks per data block)
  this->data_ = this->compute_mask_volume_->get_mask_data_block()-
>get_mask_data();

  // Bit where mask bit is stored
  this->mask_value_ = this->compute_mask_volume_->get_mask_data_block()-
>get_mask_value();
}

////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::compute_faces_setup( int num_threads )
{
  // Number of elements (cubes) in each dimension
  this->elem_nx_ = this->nx_ - 1;
  this->elem_ny_ = this->ny_ - 1;
  this->elem_nz_ = this->nz_ - 1;

  // Stores index into polygon configuration table for each element (cube)?
  // Why +1?  Maybe just padding for safety?
```

```
  this->type_buffer_.resize( ( this->nx_ + 1 ) * ( this->ny_ + 1 ) );
  // For each element (cube), holds edge ID (12 edges) back_buffer_x,
back_buffer_y,
  // front_buffer_x, front_buffer_y, and side_buffer
  this->edge_buffer_.resize( 5 );
  for ( size_t q = 0; q < 5; q++ )
  {
   this->edge_buffer_[ q ].resize( ( this->nx_ + 1 ) * ( this->ny_ + 1 ) );
  }

  // Interpolated edge points for isosurface per thread
  this->new_points_.resize( num_threads );

  // Vector of face indices per triangle, per thread
  this->new_elems_.resize( num_threads );
  // Surface areas of generated triangles per thread
  this->new_elem_areas_.resize( num_threads, 0 );
  // Offset allows zero-based indexing over multiple slices
  this->front_offset_.resize( num_threads, 0 );
  this->back_offset_.resize( num_threads, 0 );

  // Total number of isosurface points
  this->global_point_cnt_ = 0;

  this->min_point_index_.resize( this->elem_nz_ );
  this->max_point_index_.resize( this->elem_nz_ );
  this->min_face_index_.resize( this->elem_nz_ );
  this->max_face_index_.resize( this->elem_nz_ );

  this->prev_point_min_ = 0;
  this->prev_point_max_ = 0;

  this->need_abort_ = false;

}

////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::parallel_compute_faces( int thread, int
num_threads,
  boost::barrier& barrier )
{
  // Setup the algorithm and the buffers
  if ( thread == 0 ) // Only need to setup once
  {
   this->compute_faces_setup( num_threads );
  }
```

```
// An object of class barrier is a synchronization primitive used to cause a set of
threads
// to wait until they each perform a certain function or each reach a particular
point in their
// execution.  This mechanism proves useful in situations where processing
progresses in
// stages and completion of the current stage by all threads is the entry criteria for
the next
// stage.
barrier.wait();

// In order to parallelize the algorithm, each thread processes a different
horizontal strip
// of data
// Determine the element (cube) range for each thread in the y dimension
size_t ysize = static_cast< size_t >( this->ny_ / num_threads );
if ( ysize * num_threads < this->ny_ )
{
  ysize++;
}

size_t nystart = thread * ysize;
size_t nyend = ( thread + 1 ) * ysize;
if ( nyend > this->ny_ )
{
  nyend = this->ny_;
}

size_t elem_ysize = static_cast< size_t >( this->elem_ny_ / num_threads );
if ( elem_ysize * num_threads < this->elem_ny_ )
{
  elem_ysize++;
}

size_t elem_nystart = thread * elem_ysize;
size_t elem_nyend = ( thread + 1 ) * elem_ysize;
if ( elem_nyend > this->elem_ny_ )
{
  elem_nyend = this->elem_ny_;
}

// Each thread generates new points for the isosurface
PointFVector& points = this->new_points_[ thread ];
// StackVector is a SCIRun class.
// StackVector implements a subclass of the std::vector class, except that
// the vector is statically allocated on the stack for performance.
```

```
// elems = triangles
std::vector< StackVector < size_t, 3 > >& elements = this->new_elems_[ thread ];

// mark the point counter with the process number
unsigned int point_cnt = 0;

// Vector of pointers into edge_buffer_ for each of the 12 edges.  edge_buffer_
has indices into
// vector of actual points.
std::vector< unsigned int* > edge_table( 12 );

barrier.wait();

// See function description
int back_buffer_x = 0;
int back_buffer_y = 1;
int front_buffer_x = 2;
int front_buffer_y = 3;
int side_buffer = 4;

StackVector< size_t, 3 > elems( 3 );

// Get mask transform from MaskVolume
GridTransform grid_transform = this->compute_mask_volume_-
>get_grid_transform();

// Loop over all the slices
for ( size_t z = 0;  z < this->elem_nz_; z++ )
{
  // Process two adjacent slices at a time (back and front)
  // Get pointer to beginning of each slice in the data
  unsigned char* data1 = this->data_ + z * ( this->nx_ * this->ny_ );
  unsigned char* data2 = this->data_ + ( z + 1 ) * ( this->nx_ * this->ny_ );

  points.clear();
  point_cnt = thread<<24; // upper 8 bits are used to store thread id -- efficiency
trick

  // References to back/front/side tables
  UIntVector& back_edge_x = this->edge_buffer_[ back_buffer_x ];
  UIntVector& back_edge_y = this->edge_buffer_[ back_buffer_y ];

  UIntVector& front_edge_x = this->edge_buffer_[ front_buffer_x ];
  UIntVector& front_edge_y = this->edge_buffer_[ front_buffer_y ];

  UIntVector& side_edge = this->edge_buffer_[ side_buffer ];
```

```
// Use relative offsets to find edges
edge_table[ 0 ] = &( this->edge_buffer_[ back_buffer_x ][ 0 ] );
edge_table[ 1 ] = &( this->edge_buffer_[ back_buffer_y ][ 1 ] );
edge_table[ 2 ] = &( this->edge_buffer_[ back_buffer_x ][ this->nx_ ] );
edge_table[ 3 ] = &( this->edge_buffer_[ back_buffer_y ][ 0 ] );

edge_table[ 4 ] = &( this->edge_buffer_[ front_buffer_x ][ 0 ] );
edge_table[ 5 ] = &( this->edge_buffer_[ front_buffer_y ][ 1 ] );
edge_table[ 6 ] = &( this->edge_buffer_[ front_buffer_x ][ this->nx_ ] );
edge_table[ 7 ] = &( this->edge_buffer_[ front_buffer_y ][ 0 ] );

edge_table[ 8 ] = &( this->edge_buffer_[ side_buffer ][ 0 ] );
edge_table[ 9 ] = &( this->edge_buffer_[ side_buffer ][ 1 ] );
edge_table[ 10 ] = &( this->edge_buffer_[ side_buffer ][ this->nx_ ] );
edge_table[ 11 ] = &( this->edge_buffer_[ side_buffer ][ this->nx_ + 1 ] );

// Step 1: determine the type of marching cube pattern (triangles) that needs
// to go in each element (cube) and the intersecting points on each edge

// Loop over horizontal strip of elements (cubes)
for ( size_t y = nystart; y < nyend; y++ )
{
 for ( size_t x = 0; x < this->nx_; x++ )
 {
  // There are dim - 1 elements (cubes)
  if ( x < this->elem_nx_ && y < this->elem_ny_ )
  {
   // type = index into polygonal configuration table
   unsigned char type = 0;
   size_t q = y * this->nx_ + x; // Index into data
   // An 8 bit index is formed where each bit corresponds to a vertex
   // Bit on if vertex is inside surface, off otherwise
   if ( data1[ q ] & this->mask_value_ )        type |= 0x1;
   if ( data1[ q + 1 ] & this->mask_value_ )    type |= 0x2;
   if ( data1[ q + this->nx_ + 1 ] & this->mask_value_ ) type |= 0x4;
   if ( data1[ q + this->nx_ ] & this->mask_value_ )   type |= 0x8;

   if ( data2[ q ] & this->mask_value_ )        type |= 0x10;
   if ( data2[ q + 1 ] & this->mask_value_ )    type |= 0x20;
   if ( data2[ q + this->nx_ + 1 ] & this->mask_value_ ) type |= 0x40;
   if ( data2[ q + this->nx_ ] & this->mask_value_ )   type |= 0x80;

   this->type_buffer_[ q ] = type;

   // All points are inside or outside the cube -- does not contribute to the
   // isosurface
   if (type == 0x00 || type == 0xFF )
```

```
{
  continue;
}

// Since mask values are either on or off, no need to interpolate
// between vertices along edges.  Always put point in center of edge.
const float INTERP_EDGE_OFFSET_C = 0.5f;

if ( z == 0 )
{
  // top border and center ones
  if ( ( ( type>>0 ) ^ ( type>>1 ) ) & 0x01 )
  {
    PointF edge_point = PointF( static_cast< float >( x ) +
      INTERP_EDGE_OFFSET_C,
      static_cast< float >( y ),
      static_cast< float >( z ) );

    // Transform point by mask transform
    edge_point = grid_transform.project( edge_point );

    points.push_back( edge_point );
    back_edge_x[ q ] = point_cnt;
    point_cnt++;
  }

  // bottom border one
  if ( ( y == this->elem_ny_ - 1 ) && ( ( ( type>>2 ) ^ ( type>>3 ) ) & 0x01 ) )
  {
    PointF edge_point = PointF( static_cast< float >( x ) +
      INTERP_EDGE_OFFSET_C,
      static_cast< float>( y + 1 ),
      static_cast< float >( z ) );

    // Transform point by mask transform
    edge_point = grid_transform.project( edge_point );

    points.push_back( edge_point );
    back_edge_x[ q + this->nx_ ] = point_cnt;
    point_cnt++;
  }

  // left border and center ones
  if ( ( ( type>>0 ) ^ ( type>>3 ) ) & 0x01 )
  {
    PointF edge_point = PointF( static_cast< float >( x ),
      static_cast< float >( y ) + INTERP_EDGE_OFFSET_C,
```

```
            static_cast< float >( z ) );

          // Transform point by mask transform
          edge_point = grid_transform.project( edge_point );

          points.push_back( edge_point );
          back_edge_y[ q ] = point_cnt;
          point_cnt++;
        }

        // right one
        if ( ( x == this->elem_nx_ - 1 ) && ( ( ( type>>1 ) ^ ( type>>2 ) ) & 0x01 ) )
        {
          PointF edge_point = PointF( static_cast< float >( x + 1 ),
            static_cast< float >( y ) + INTERP_EDGE_OFFSET_C,
            static_cast< float >( z ) );

          // Transform point by mask transform
          edge_point = grid_transform.project( edge_point );

          points.push_back( edge_point );
          back_edge_y[ q + 1 ] = point_cnt;
          point_cnt++;
        }
      }

      // top border and center ones
      if ( ( ( type>>4 ) ^ ( type>>5 ) ) & 0x01 )
      {
        PointF edge_point = PointF( static_cast< float >( x ) +
INTERP_EDGE_OFFSET_C,
          static_cast< float >( y ),
          static_cast< float >( z + 1 ) );

        // Transform point by mask transform
        edge_point = grid_transform.project( edge_point );

        points.push_back( edge_point );
        front_edge_x[ q ] = point_cnt;
        point_cnt++;
      }

      // bottom border one

      if ( ( y == this->elem_ny_ - 1 ) && ( ( ( type>>6 ) ^ ( type>>7 ) ) & 0x01 ) )
      {
        PointF edge_point = PointF( static_cast< float >( x ) +
```

```
INTERP_EDGE_OFFSET_C,
          static_cast< float >( y + 1 ),
          static_cast< float >( z + 1 ) );

        // Transform point by mask transform
        edge_point = grid_transform.project( edge_point );

        points.push_back( edge_point );
        front_edge_x[ q + this->nx_ ] = point_cnt;
        point_cnt++;
      }

      // left border and center ones
      if ( ( ( type>>4 ) ^ ( type>>7 ) ) & 0x01 )
      {
        PointF edge_point = PointF( static_cast< float >( x ),
          static_cast< float >( y ) + INTERP_EDGE_OFFSET_C,
          static_cast< float >( z + 1 ) );

        // Transform point by mask transform
        edge_point = grid_transform.project( edge_point );

        points.push_back( edge_point );
        front_edge_y[ q ] = point_cnt;
        point_cnt++;
      }

      // bottom one
      if ( ( x== this->elem_nx_ - 1 ) && ( ( ( type>>5 ) ^ ( type>>6 ) ) & 0x01 ) )
      {
        PointF edge_point = PointF( static_cast< float >( x + 1 ),
          static_cast< float >( y ) + INTERP_EDGE_OFFSET_C,
          static_cast< float >( z + 1 ) );

        // Transform point by mask transform
        edge_point = grid_transform.project( edge_point );

        points.push_back( edge_point );
        front_edge_y[ q + 1 ] = point_cnt;
        point_cnt++;
      }

      // side edges

      if ( ( ( type>>0 ) ^ ( type >> 4 ) ) & 0x01 )
      {
        PointF edge_point = PointF( static_cast< float >( x ),
```

```
            static_cast< float >( y ),
            static_cast< float >( z ) + INTERP_EDGE_OFFSET_C );

          // Transform point by mask transform
          edge_point = grid_transform.project( edge_point );

          points.push_back( edge_point );
          side_edge[ q ] = point_cnt;
          point_cnt++;
        }


        if ( ( x == this->elem_nx_ - 1 ) && ( ( ( type>>1 ) ^ ( type>>5 ) ) & 0x01 ) )
        {
          PointF edge_point = PointF( static_cast< float >( x + 1 ),
            static_cast< float >( y ),
            static_cast< float >( z ) + INTERP_EDGE_OFFSET_C );

          // Transform point by mask transform
          edge_point = grid_transform.project( edge_point );

          points.push_back( edge_point );
          side_edge[ q + 1 ] = point_cnt;
          point_cnt++;
        }

        if ( ( y == this->elem_ny_ - 1 ) && ( ( ( type>>3 ) ^ ( type>>7 ) ) & 0x01 ) )
        {
          PointF edge_point = PointF( static_cast< float >( x ),
            static_cast< float >( y + 1 ),
            static_cast< float >( z ) + INTERP_EDGE_OFFSET_C );

          // Transform point by mask transform
          edge_point = grid_transform.project( edge_point );

          points.push_back( edge_point );
          side_edge[ q + this->nx_ ] = point_cnt;
          point_cnt++;
        }

        if ( ( ( y == this->elem_ny_ - 1 ) && ( x == this->elem_nx_ - 1 ) ) &&
          ( ( ( type>>2 ) ^ ( type>>6 ) ) & 0x01 ) )
        {
          PointF edge_point = PointF( static_cast< float >( x + 1 ),
            static_cast< float >( y + 1 ),
            static_cast< float >( z ) + INTERP_EDGE_OFFSET_C );
```

```
            // Transform point by mask transform
            edge_point = grid_transform.project( edge_point );

            points.push_back( edge_point );
            side_edge[ q + this->nx_ + 1 ] = point_cnt;
            point_cnt++;
          }
        }
      }
    }

    barrier.wait();
    elements.clear();

    // Combine points from all threads
    if ( thread == 0 )
    {
      size_t local_size = 0;
      for ( int p = 0; p < num_threads; p++ )
      {
        this->front_offset_[ p ] = this->global_point_cnt_ + local_size;
        if ( z == 0 )
        {
          this->back_offset_[ p ] = this->front_offset_[ p ];
        }
        local_size += this->new_points_[ p ].size();
        PointFVector& points = this->new_points_[ p ];
        for ( size_t q = 0; q < points.size(); q++ )
        {
          this->points_.push_back( points[ q ] );
        }
      }
      this->global_point_cnt_ += local_size;

      this->min_point_index_[ z ] = this->prev_point_min_;
      this->max_point_index_[ z ] = static_cast<unsigned int>( this->points_.size() );
      this->prev_point_min_ = this->prev_point_max_;
      this->prev_point_max_ = this->max_point_index_[ z ];
    }

    barrier.wait();

    // Build triangles
    for ( size_t y = elem_nystart; y < elem_nyend; y++ )
    {
      for (size_t x=0;x<elem_nx_;x++)
      {
```

```
        size_t elem_offset = y * this->nx_ + x;
        unsigned char type = this->type_buffer_[ elem_offset ];

        // All points are inside or outside the cube -- does not contribute to the
        // isosurface
        if ( type == 0 || type == 0xFF )
        {
          continue;
        }

        // Get the edges from the marching cube table
        const MarchingCubesTableType& table = MARCHING_CUBES_TABLE_C[
    type ];

        for ( int k = 0; k < table.num_triangles_; k++ )
        {
          // Get the edge index (0-11 for 12 edges)
          int i1 = table.edges_[ 3 * k ];
          int i2 = table.edges_[ 3 * k + 1 ];
          int i3 = table.edges_[ 3 * k + 2 ];

          unsigned int p1 = edge_table[ i1 ][ elem_offset ];
          unsigned int p2 = edge_table[ i2 ][ elem_offset ];
          unsigned int p3 = edge_table[ i3 ][ elem_offset ];

          if ( i1 < 4 )
          {
            elems[ 0 ] = ( p1 & 0x00FFFFFF ) + this->back_offset_[ p1>>24 ];
          }
          else
          {
            elems[ 0 ] = ( p1 & 0x00FFFFFF ) + this->front_offset_[ p1>>24 ];
          }

          if ( i2 < 4 )
          {
            elems[ 1 ] = ( p2 & 0x00FFFFFF ) + this->back_offset_[ p2>>24 ];
          }
          else
          {
            elems[ 1 ] = ( p2 & 0x00FFFFFF ) + this->front_offset_[ p2>>24 ];
          }

          if ( i3 < 4 )
          {
            elems[ 2 ] = ( p3 & 0x00FFFFFF ) + this->back_offset_[ p3>>24 ];
          }
```

```
        else
        {
          elems[ 2 ] = ( p3 & 0x00FFFFFF ) + this->front_offset_[ p3>>24 ];
        }
        elements.push_back( elems );
        // Add the area of the triangle to the total
        this->new_elem_areas_[ thread ] += 0.5f *
          Cross( this->points_[ elems[ 1 ] ] - this->points_[ elems[ 0 ] ],
          this->points_[ elems[ 2 ] ] - this->points_[ elems[ 0 ] ] ).length();
      }
    }
  }

  std::swap( back_buffer_x, front_buffer_x );
  std::swap( back_buffer_y, front_buffer_y );
  barrier.wait();

  if ( thread == 0 )
  {
    this->min_face_index_[ z ] = static_cast<unsigned int>( this->faces_.size() );

    for ( int w = 0;  w < num_threads; w++ )
    {
      std::vector< StackVector< size_t, 3 > >& pelements = this->new_elems_[ w ];
      for ( size_t p = 0; p < pelements.size(); p++ )
      {
        StackVector< size_t, 3 >& el = pelements[ p ];
        this->faces_.push_back( static_cast< unsigned int >( el[ 0 ] ) );
        this->faces_.push_back( static_cast< unsigned int >( el[ 1 ] ) );
        this->faces_.push_back( static_cast< unsigned int >( el[ 2 ] ) );
      }
    }
    this->back_offset_ = this->front_offset_;

    this->max_face_index_[ z ] = static_cast<unsigned int>( this->faces_.size() );
  }

  if ( thread == 0 )
  {
    if ( this->check_abort_() )
    {
      this->need_abort_ = true;
    }
  }

  barrier.wait();
```

```
    if ( this->need_abort_ )
    {
      return;
    }

    // Update progress based on number of z slices processed
    double compute_progress =
      static_cast< double >( z + 1 ) / static_cast< double >( this->elem_nz_ );
    double total_progress = compute_progress *
COMPUTE_PERCENT_PROGRESS_C;
    this->isosurface_->update_progress_signal_( total_progress );
   }

   barrier.wait();

   // Add up surface areas computed in all threads
   if ( thread == 0 )
   {
    for ( int p = 0; p < num_threads; ++p )
    {
     this->area_ += this->new_elem_areas_[ p ];
    }
   }

   barrier.wait();
}

//////////////////////////////////////////////////////////////////////////

// Translates border face coords (i, j) to volume coords (x, y, z).

void HDX_volumeToIsosurface::translate_cap_coords( int cap_num, float i, float j,
  float& x, float& y, float& z )
{
  switch( cap_num )
  {
    case 0: // x, y, z = 0 (front)
      x = i;
      y = j;
      z = 0;
      break;
    case 1: // x, z, y = 0  (top)
      x = i;
      y = 0;
      z = j;
      break;
    case 2: // y, z, x = 0 (left)
```

```
        x = 0;
        y = i;
        z = j;
        break;
      case 3: // x, y, z = nz - 1 (back)
        x = i;
        y = j;
        z = static_cast< float >( this->nz_ - 1 );
        break;
      case 4: // x, z, y = ny - 1 (bottom)
        x = i;
        y = static_cast< float >( this->ny_ - 1 );
        z = j;
        break;
      case 5: // y, z, x = nx - 1 (right)
        x = static_cast< float >( this->nx_ - 1 );
        y = i;
        z = j;
        break;
      default:
        x = y = z = 0;
        break;
    }
  }

  ////////////////////////////////////////////////////////////////////////

  size_t HDX_volumeToIsosurface::get_data_index( float x, float y, float z )
  {
    size_t data_index = ( static_cast< size_t >( z ) * this->nx_ * this->ny_ ) +
      ( static_cast< size_t >( y ) * this->nx_ ) + static_cast< size_t >( x );
    return data_index;
  }

  ////////////////////////////////////////////////////////////////////////

  void HDX_volumeToIsosurface::compute_cap_faces()
  {

    size_t nx = this->nx_; // Store local copy just to make code more concise
    size_t ny = this->ny_;
    size_t nz = this->nz_;
    std::vector< std::pair< size_t, size_t > > cap_dimensions;
    cap_dimensions.push_back( std::make_pair( nx, ny ) ); // front and back caps
    cap_dimensions.push_back( std::make_pair( nx, nz ) ); // top and bottom caps
    cap_dimensions.push_back( std::make_pair( ny, nz ) ); // left and right side caps
    PointF elem_vertices[ 3 ]; // Temporary storage for triangle vertices
```

```
// For each of 6 caps
for( int cap_num = 0; cap_num < 6; cap_num++ )
{
  unsigned int min_point_index = static_cast< unsigned int >( this->points_.size()
);
  unsigned int min_face_index = static_cast< unsigned int >( this->faces_.size() );

  // Each
  // STEP 1: Find cell types

  // Calculate ni, nj for this face
  size_t ni = cap_dimensions[ cap_num % 3 ].first;
  size_t nj = cap_dimensions[ cap_num % 3 ].second;

  // Since each cell has 4 nodes, we only need a char to represent the type.  Only
using bits
  // 0 - 3.
  // Create an array of type per index
  size_t num_cells = ( ni - 1 )* ( nj - 1 );
  UCharVector cell_types( num_cells );
  // Loop through all 2D cells to find each cell type
  size_t cell_index = 0;
  bool some_nodes_on = false;
  for( float j = 0; j < nj - 1; j++ )
  {
    for( float i = 0; i < ni - 1; i++ )
    {
      // Build type
      // type = index into polygonal configuration table
      unsigned char cell_type = 0;

      // A 4 bit index is formed where each bit corresponds to a vertex
      // Bit on if vertex is inside surface, off otherwise
      float x, y, z;
      this->translate_cap_coords( cap_num, i, j, x, y, z );
      size_t data_index = this->get_data_index( x, y, z );
      if ( this->data_[ data_index ] & this->mask_value_ )  cell_type |= 0x1;

      this->translate_cap_coords( cap_num, i + 1, j, x, y, z );
      data_index = this->get_data_index( x, y, z );
      if ( this->data_[ data_index ] & this->mask_value_ )  cell_type |= 0x2;

      this->translate_cap_coords( cap_num, i + 1, j + 1, x, y, z );
      data_index = this->get_data_index( x, y, z );
      if ( this->data_[ data_index ] & this->mask_value_ )  cell_type |= 0x4;
```

```
        this->translate_cap_coords( cap_num, i, j + 1, x, y, z );
        data_index = this->get_data_index( x, y, z );
        if ( this->data_[ data_index ] & this->mask_value_ )  cell_type |= 0x8;

        cell_types[ cell_index ] = cell_type;
        cell_index++;

        if( cell_type != 0 )
        {
          some_nodes_on = true;
        }
      }
    }

    // If no border nodes for this cap are on, skip this cap
    if( !some_nodes_on )
    {
      continue;
    }

    // STEP 2: Add nodes to points list and translation table

    // Create translation table (2D matrix storing indices into actual points vector)
    // size_t point_trans_table[num cells = (nx - 1) * (ny - 1)][8 canonical indices (4
nodes, 4 edge points)]
    std::vector< boost::array< unsigned int, 8 > > point_trans_table( num_cells );

    // Get mask transform from MaskVolume
    GridTransform grid_transform = this->compute_mask_volume_-
>get_grid_transform();

    // All border nodes that are "on" are included as points because they are all
adjacent to
    // the imaginary padding that is "off."
    // Loop through all nodes

    for( float j = 0; j < nj; j++ )
    {
      for( float i = 0; i < ni; i++ )
      {
        // Translate (i, j) to (x, y, z) for this cap
        float x, y, z;
        this->translate_cap_coords( cap_num, i, j, x, y, z );

        // Look up mask value at (x, y, z)
        size_t data_index = this->get_data_index( x, y, z );
```

```
        // If mask value on
        if ( this->data_[ data_index ] & this->mask_value_ )
        {
          // Transform point by mask transform
          PointF node_point = grid_transform.project( PointF( x, y, z ) );
          // Add node to the points list.
          this->points_.push_back( node_point );
          unsigned int point_index =
            static_cast< unsigned int >( this->points_.size() - 1 );

          // Add relevant canonical coordinates to translation table for adjacent cells.
          // Find indices and canonical coordinates of 1-4 adjacent cells

          // Lower right cell index
          if( i < ni - 1 && j < nj - 1 )
          {
            size_t cell_index = static_cast< size_t >( ( j * ( ni - 1 ) ) + i );
            size_t canonical_coordinate = 0;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }

          // Lower left cell index
          if( i > 0 && j < nj - 1 )
          {
            size_t cell_index = static_cast< size_t >( ( j * ( ni - 1 ) ) + i - 1 );
            size_t canonical_coordinate = 1;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }

          // Upper left cell index
          if( i > 0 && j > 0 )
          {
            size_t cell_index =
              static_cast< size_t >( ( ( j - 1 ) * ( ni - 1 ) ) + i - 1 );
            size_t canonical_coordinate = 2;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }

          // Upper right cell index
          if( i < ni - 1 && j > 0 )
          {
            size_t cell_index = static_cast< size_t >( ( ( j - 1 ) * ( ni - 1 ) ) + i );
            size_t canonical_coordinate = 3;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }
        }
      }
```

```
    }

    // STEP 3: Find edge nodes, add to points list and translation table

    // Check vertical edges
    for( float j = 0; j < nj - 1; j++ )
    {
      for( float i = 0; i < ni; i++ )
      {
        // Find endpoint nodes
        float start_x, start_y, start_z;
        this->translate_cap_coords( cap_num, i, j, start_x, start_y, start_z );
        float end_x, end_y, end_z;
        this->translate_cap_coords( cap_num, i, j + 1, end_x, end_y, end_z );

        size_t start_data_index = this->get_data_index( start_x, start_y, start_z );
        size_t end_data_index = this->get_data_index( end_x, end_y, end_z );

        // If edge is "split" (one endpoint node is on and the other is off)
        unsigned char start_bit = this->data_[ start_data_index ] & this-
>mask_value_;
        unsigned char end_bit = this->data_[ end_data_index ] & this->mask_value_;
        if ( start_bit != end_bit )
        {
          // Find edge point
          float edge_x, edge_y, edge_z;
          this->translate_cap_coords( cap_num, i, j + 0.5f,
            edge_x, edge_y, edge_z );

          // Transform point by mask transform
          PointF edge_point = grid_transform.project( PointF( edge_x, edge_y,
edge_z) );
          // Add edge to the points list.
          this->points_.push_back( edge_point );
          unsigned int point_index =
            static_cast< unsigned int >( this->points_.size() - 1 );

          // Add the relevant canonical coordinates to the translation table for adjacent
1-2 cells.

          // Left cell index
          if( i > 0 )
          {
            size_t cell_index = static_cast< size_t >( ( j * ( ni - 1 ) ) + i - 1 );
            size_t canonical_coordinate = 5;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }
```

```
        // Right cell index
        if( i < ni - 1 )
        {
          size_t cell_index = static_cast< size_t >( ( j * ( ni - 1 ) ) + i );
          size_t canonical_coordinate = 7;
          point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
        }
      }
    }
  }

  // Check horizontal edges
  for( float j = 0; j < nj; j++ )
  {
    for( float i = 0; i < ni - 1; i++ )
    {
      // Find endpoint nodes
      float start_x, start_y, start_z;
      this->translate_cap_coords( cap_num, i, j, start_x, start_y, start_z );
      float end_x, end_y, end_z;
      this->translate_cap_coords( cap_num, i + 1, j, end_x, end_y, end_z );

      size_t start_data_index = this->get_data_index( start_x, start_y, start_z );
      size_t end_data_index = this->get_data_index( end_x, end_y, end_z );

      // If edge is "split" (one endpoint node is on and the other is off)
      unsigned char start_bit = this->data_[ start_data_index ] & this-
>mask_value_;
      unsigned char end_bit = this->data_[ end_data_index ] & this->mask_value_;
      if ( start_bit != end_bit )
      {
        // Find edge point
        float edge_x, edge_y, edge_z;
        this->translate_cap_coords( cap_num, i + 0.5f, j,
          edge_x, edge_y, edge_z );

        // Transform point by mask transform
        PointF edge_point = grid_transform.project( PointF( edge_x, edge_y,
edge_z) );
        // Add edge to the points list.
        this->points_.push_back( edge_point );
        unsigned int point_index =
          static_cast< unsigned int >( this->points_.size() - 1 );

        // Add the relevant canonical coordinates to the translation table for adjacent
1-2 cells.
```

```
          // Upper cell index
          if( j > 0 )
          {
            size_t cell_index = static_cast< size_t >( ( ( j - 1 ) * ( ni - 1 ) ) + i );
            size_t canonical_coordinate = 6;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }

          // Lower cell index
          if( j < nj - 1 )
          {
            size_t cell_index = static_cast< size_t >( ( j * ( ni - 1 ) ) + i );
            size_t canonical_coordinate = 4;
            point_trans_table[ cell_index ][ canonical_coordinate ] = point_index;
          }
        }
      }
    }

    // STEP 4: Create face geometry (points + triangle indices)

    // Add points list to existing vertex list
    // For each cell
    for( size_t cell_index = 0; cell_index < num_cells; cell_index++ )
    {
      // Lookup cell type (already stored in a 1D vector)
      unsigned char cell_type = cell_types[ cell_index ];

      // Couldn't I just look up type on the fly here?  Or does that make
parallelization harder?
      // Look up the facet combo for this type
      const CappingTableType& tesselation = CAPPING_TABLE_C[ cell_type ];

      // For each triangle in the tesselation for this type
      for( int triangle_index = 0; triangle_index < tesselation.num_triangles_;
triangle_index++ )
      {
        // For each vertex in the triangle
        for( int triangle_point_index = 0; triangle_point_index < 3;
triangle_point_index++ )
        {
          // Find canonical index for the point
          int canonical_index =
            tesselation.points_[ 3 * triangle_index + triangle_point_index ];

          // Look up the point index in the translation table for this cell
```

```
          unsigned int point_index = point_trans_table[ cell_index ][ canonical_index ];
          // Store the point coordinates in the temporary variable
          elem_vertices[ triangle_point_index ] = this->points_[ point_index ];
          // Add point index to the faces list
          this->faces_.push_back( point_index );
        }
        // Compute the area of  the triangle and add it to the total area
        this->area_ += 0.5f * Cross( elem_vertices[ 1 ] - elem_vertices[ 0 ],
          elem_vertices[ 2 ] - elem_vertices[ 0 ] ).length();
      }
    }

    // Create a rendering "patch" for each cap
    unsigned int max_point_index = static_cast< unsigned int >( this->points_.size()
);
    unsigned int max_face_index = static_cast< unsigned int >( this->faces_.size()
);
    if( min_point_index != max_point_index && min_face_index != max_face_index
)
    {
      this->min_point_index_.push_back( min_point_index );
      this->max_point_index_.push_back( max_point_index );

      this->min_face_index_.push_back( min_face_index );
      this->max_face_index_.push_back( max_face_index );
    }
  }
}

////////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::parallel_compute_normals( int thread, int
num_threads,
  boost::barrier& barrier )
{
  size_t num_vertices = this->points_.size();

  if ( thread == 0 ) // Only need to setup once
  {
    // Reset the normals vector
    this->normals_.clear();
    this->normals_.resize( num_vertices, VectorF( 0, 0, 0 ) );
  }

  // All threads have to wait until setup is done before proceeding
  barrier.wait();
```

```
  // In order to parallelize the algorithm, each thread processes a different range of
vertex
  // indices [vertex_index_start, vertex_index_end)
  size_t vertex_range_size = static_cast< size_t >( num_vertices / num_threads );
  if ( vertex_range_size * num_threads < num_vertices )
  {
    vertex_range_size++;
  }

  size_t vertex_index_start = thread * vertex_range_size;
  size_t vertex_index_end = ( thread + 1 ) * vertex_range_size;
  if ( vertex_index_end > num_vertices )
  {
    vertex_index_end = num_vertices;
  }

  // For each face
  for( size_t i = 0; i + 2 < this->faces_.size(); i += 3 )
  {
    size_t vertex_index1 = this->faces_[ i ];
    size_t vertex_index2 = this->faces_[ i + 1 ];
    size_t vertex_index3 = this->faces_[ i + 2 ];

    // If this face has at least one vertex in our range
    if( ( vertex_index_start <= vertex_index1 && vertex_index1 < vertex_index_end
) ||
      ( vertex_index_start <= vertex_index2 && vertex_index2 < vertex_index_end )
||
      ( vertex_index_start <= vertex_index3 && vertex_index3 < vertex_index_end )
)
    {
      // Get vertices of face
      PointF p1 = this->points_[ vertex_index1 ];
      PointF p2 = this->points_[ vertex_index2 ];
      PointF p3 = this->points_[ vertex_index3 ];

      // Calculate cross product of edges
      VectorF v0 = p3 - p2;
      VectorF v1 = p1 - p2;
      VectorF n = Cross( v0, v1 );

      // Add to normal for each vertex in our range
      if( ( vertex_index_start <= vertex_index1 && vertex_index1 <
vertex_index_end ) )
      {
        this->normals_[ vertex_index1 ] += n;
      }
```

```
      if( ( vertex_index_start <= vertex_index2 && vertex_index2 <
vertex_index_end ) )
      {
        this->normals_[ vertex_index2 ] += n;
      }
      if( ( vertex_index_start <= vertex_index3 && vertex_index3 <
vertex_index_end ) )
      {
        this->normals_[ vertex_index3 ] += n;
      }
    }
  }

  // For each vertex in our range
  for( size_t i = vertex_index_start; i < vertex_index_end; i++ )
  {
    // Normalize normal
    this->normals_[ i ].normalize();
  }
}

////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::upload_to_vertex_buffer()
{
  if ( !this->surface_changed_ && !this->values_changed_ )
  {
    return;
  }

  size_t num_of_parts = this->part_points_.size();
  bool has_values = this->values_.size() == this->points_.size();

  // Estimate the size of video memory required to upload the isosurface
  ptrdiff_t total_size = 0;
  for ( size_t i = 0; i < num_of_parts; ++i )
  {
    unsigned int num_pts = this->part_points_[ i ].second - this->part_points_[ i
].first;
    ptrdiff_t vertex_size = num_pts * sizeof( PointF );
    ptrdiff_t normal_size = num_pts * sizeof( VectorF );
    ptrdiff_t value_size = has_values ? num_pts * sizeof( float ) : 0;
    unsigned int num_face_indices = this->part_faces_[ i ].second - this-
>part_faces_[ i ].first;
    ptrdiff_t face_size = num_face_indices * sizeof( unsigned int );
    ptrdiff_t batch_size = vertex_size + normal_size + value_size + face_size;
    CORE_LOG_MESSAGE( "Isosurface Batch " + ExportToString( i ) + ": " +
```

```
            ExportToString( num_pts ) + " vertices, " +
            ExportToString( num_face_indices / 3 ) + " triangles. Total memory: " +
            ExportToString( batch_size ) );
        total_size += batch_size;
    }
    CORE_LOG_MESSAGE( "Total memory required for the isosurface: " +
          ExportToString( total_size ) );

    if ( total_size + ( 20 << 20 ) > static_cast< ptrdiff_t >(
      RenderResources::Instance()->get_vram_size() ) )
    {
      CORE_LOG_WARNING( "Could not fit the isosurface in GPU memory" );
      this->vbo_batches_.clear();
      this->surface_changed_ = false;
      this->values_changed_ = false;
      this->vbo_available_ = false;
      return;
    }

    RenderResources::lock_type rr_lock( RenderResources::GetMutex() );
    this->vbo_batches_.resize( num_of_parts );
    for ( size_t i = 0; i < num_of_parts; ++i )
    {
      this->vbo_batches_[ i ].reset( new VertexBufferBatch );
      this->vbo_batches_[ i ]->vertex_buffer_.reset( new
Core::VertexAttribArrayBuffer );
      this->vbo_batches_[ i ]->normal_buffer_.reset( new
Core::VertexAttribArrayBuffer );
      this->vbo_batches_[ i ]->faces_buffer_.reset( new Core::ElementArrayBuffer );
      this->vbo_batches_[ i ]->vertex_buffer_->set_array(
        VertexAttribArrayType::VERTEX_E, 3, GL_FLOAT, 0, 0 );
      this->vbo_batches_[ i ]->normal_buffer_->set_array(
        VertexAttribArrayType::NORMAL_E, GL_FLOAT, 0, 0 );

      unsigned int num_pts = this->part_points_[ i ].second - this->part_points_[ i
].first;
      ptrdiff_t vertex_size = num_pts * sizeof( PointF );
      ptrdiff_t normal_size = num_pts * sizeof( VectorF );
      unsigned int num_face_indices = this->part_faces_[ i ].second - this-
>part_faces_[ i ].first;
      ptrdiff_t face_size = num_face_indices * sizeof( unsigned int );

      this->vbo_batches_[ i ]->vertex_buffer_->set_buffer_data( vertex_size,
        &this->points_[ this->part_points_[ i ].first ], GL_STATIC_DRAW );
      this->vbo_batches_[ i ]->normal_buffer_->set_buffer_data( normal_size,
        &this->normals_[ this->part_points_[ i ].first ], GL_STATIC_DRAW );
      this->vbo_batches_[ i ]->faces_buffer_->set_buffer_data( face_size,
```

```
      &this->part_indices_[ i ][ 0 ], GL_STATIC_DRAW );
    if ( has_values )
    {
      this->vbo_batches_[ i ]->value_buffer_.reset( new
Core::VertexAttribArrayBuffer );
      this->vbo_batches_[ i ]->value_buffer_->set_generic_array( 1, 1, GL_FLOAT,
        GL_FALSE, 0, 0 );
      this->vbo_batches_[ i ]->value_buffer_->set_buffer_data( num_pts * sizeof(
float ),
        &this->values_[ this->part_points_[ i ].first ], GL_STATIC_DRAW );
    }
   }

  this->surface_changed_ = false;
  this->values_changed_ = false;
  this->vbo_available_ = true;
}

////////////////////////////////////////////////////////////////////////

void HDX_volumeToIsosurface::reset()
{
  this->points_.clear();
  this->normals_.clear();
  this->faces_.clear();
  this->values_.clear();
}

////////////////////////////////////////////////////////////////////////

Isosurface::Isosurface( const MaskVolumeHandle& mask_volume ) :
  private_( new HDX_volumeToIsosurface )
{
  this->private_->isosurface_ = this;
  this->private_->orig_mask_volume_ = mask_volume;
  this->private_->compute_mask_volume_ = mask_volume;
  this->private_->surface_changed_ = false;
  this->private_->values_changed_ = false;
  this->private_->vbo_available_ = false;

}

////////////////////////////////////////////////////////////////////////

void Isosurface::compute( double quality_factor, bool capping_enabled,
  boost::function< bool () > check_abort )
{
```

```
  lock_type lock( this->get_mutex() );

  this->private_->points_.clear();
  this->private_->normals_.clear();
  this->private_->faces_.clear();
  this->private_->values_.clear();
  this->private_->area_ = 0;
  this->private_->values_changed_ = false;
  this->private_->check_abort_ = check_abort;

  {
    Core::MaskVolume::shared_lock_type vol_lock( this->private_-
>orig_mask_volume_->get_mutex() );

    // Initially assume we're computing the isosurface for the original volume (not
downsampled)
    this->private_->compute_mask_volume_ = this->private_->orig_mask_volume_;

    // Downsample mask if needed
    if( quality_factor != 1.0 )
    {
      assert( quality_factor == 0.5 || quality_factor == 0.25 || quality_factor == 0.125
);
      Parallel parallel_downsample( boost::bind(
&HDX_volumeToIsosurface::parallel_downsample_mask,
        this->private_, _1, _2, _3, quality_factor ) );
      parallel_downsample.run();
    }

    if ( check_abort() )
    {
      // leave it in a decent state
      this->private_->reset();
      return;
    }

    // Copy values to members just to simplify and shorten code.
    this->private_->compute_setup();

    // Compute isosurface without caps
    Parallel parallel_faces( boost::bind(
&HDX_volumeToIsosurface::parallel_compute_faces,
      this->private_, _1, _2, _3 ) );
    parallel_faces.run();

    if ( check_abort() )
    {
```

```
      // leave it in a decent state
      this->private_->reset();
      return;
    }

    // Compute isosurface caps
    if( capping_enabled )
    {
      this->private_->compute_cap_faces();
    }
  }

  // Check for empty isosurface
  if( this->private_->points_.size() == 0 )
  {
    return;
  }

  Parallel parallel_normals( boost::bind(
&HDX_volumeToIsosurface::parallel_compute_normals,
    this->private_, _1, _2, _3 ) );
  parallel_normals.run();

  if ( check_abort() )
  {
    // leave it in a decent state
    this->private_->reset();
    return;
  }

  this->update_progress_signal_(
HDX_volumeToIsosurface::COMPUTE_PERCENT_PROGRESS_C +
    HDX_volumeToIsosurface::NORMAL_PERCENT_PROGRESS_C );

  this->private_->type_buffer_.clear();
  this->private_->edge_buffer_.clear();
  this->private_->new_points_.clear();
  this->private_->new_elems_.clear();
  this->private_->new_elem_areas_.clear();
  this->private_->front_offset_.clear();
  this->private_->back_offset_.clear();

  this->private_->part_points_.clear();
  this->private_->part_faces_.clear();
  this->private_->part_indices_.clear();

  unsigned int num_faces = 0;
```

```
  unsigned int min_point_index = 0;
  unsigned int min_face_index = 0;

  for ( size_t j = 0; j < this->private_->min_point_index_.size(); j++ )
  {
    if ( num_faces == 0 )
    {
      min_point_index = this->private_->min_point_index_[ j ];
      min_face_index = this->private_->min_face_index_[ j ];
    }

    num_faces += this->private_->max_face_index_[ j ] - this->private_-
>min_face_index_[ j ];

    if ( num_faces > 0 && (num_faces > 1000000 || j == this->private_-
>min_point_index_.size() - 1 ) )
    {
      this->private_->part_points_.push_back( std::make_pair(
        min_point_index, this->private_->max_point_index_[ j ] ) );
      this->private_->part_faces_.push_back( std::make_pair(
        min_face_index, this->private_->max_face_index_[ j ] ) );
      num_faces = 0;
    }

    if ( check_abort() )
    {
      // leave it in a decent state
      this->private_->reset();
      return;
    }

    // Update progress
    double partition_progress =
      static_cast< double >( j + 1 ) / static_cast< double >( this->private_-
>min_point_index_.size() );
    double total_progress =
HDX_volumeToIsosurface::COMPUTE_PERCENT_PROGRESS_C +
      HDX_volumeToIsosurface::NORMAL_PERCENT_PROGRESS_C +
      ( partition_progress *
HDX_volumeToIsosurface::PARTITION_PERCENT_PROGRESS_C );
    this->update_progress_signal_( total_progress );
  }
  size_t num_batches = this->private_->part_points_.size();
  for ( size_t i = 0; i < num_batches; ++i )
  {
    unsigned int num_face_indices = this->private_->part_faces_[ i ].second -
      this->private_->part_faces_[ i ].first;
```

```
    UIntVector local_indices( num_face_indices );
    for ( size_t j = 0; j < num_face_indices; ++j )
    {
      size_t pt_global_idx = this->private_->part_faces_[ i ].first + j;
      local_indices[ j ] = this->private_->faces_[ pt_global_idx ] -
        this->private_->part_points_[ i ].first;
      assert( local_indices[ j ] >= 0 &&
          local_indices[ j ] < ( this->private_->part_points_[ i ].second -
                this->private_->part_points_[ i ].first ) );
    }
    this->private_->part_indices_.push_back( local_indices );
  }

  this->private_->min_point_index_.clear();
  this->private_->max_point_index_.clear();
  this->private_->min_face_index_.clear();
  this->private_->max_face_index_.clear();

  this->private_->surface_changed_ = true;

  if ( check_abort() )
  {
    // leave it in a decent state
    this->private_->reset();
    return;
  }

  this->update_progress_signal_( 1.0 );

}

//////////////////////////////////////////////////////////////////////////

const PointFVector& Isosurface::get_points() const
{
  return this->private_->points_;
}

//////////////////////////////////////////////////////////////////////////

const UIntVector& Isosurface::get_faces() const
{
  return this->private_->faces_;
}

//////////////////////////////////////////////////////////////////////////
```

```
const VectorFVector& Isosurface::get_normals() const
{
  return this->private_->normals_;
}

////////////////////////////////////////////////////////////////////////

const FloatVector& Isosurface::get_values() const
{
  return this->private_->values_;
}

////////////////////////////////////////////////////////////////////////

bool Isosurface::set_values( const FloatVector& values )
{
  if( !( values.size() == this->private_->points_.size() || values.size() == 0 ) )
  {
    return false;
  }
  this->private_->values_ = values;
  this->private_->values_changed_ = true;
  return true;
}

////////////////////////////////////////////////////////////////////////

void Isosurface::set_color_map( ColorMapHandle color_map )
{
  lock_type lock( this->get_mutex() );
  this->private_->color_map_ = color_map;
}

////////////////////////////////////////////////////////////////////////

Core::ColorMapHandle Isosurface::get_color_map() const
{
  lock_type lock( this->get_mutex() );
  return this->private_->color_map_;
}

////////////////////////////////////////////////////////////////////////

void Isosurface::redraw( bool use_colormap )
{
  lock_type lock( this->get_mutex() );
```

```
  // Check for empty isosurface
  if( this->private_->points_.size() == 0 )
  {
    return;
  }

  this->private_->upload_to_vertex_buffer();
  size_t num_batches = this->private_->part_points_.size();
  bool has_values = this->private_->values_.size() == this->private_-
>points_.size();

  // Use the uploaded VBO for rendering if it's available
  if ( this->private_->vbo_available_ )
  {
    for ( size_t i = 0; i < num_batches; ++i )
    {
      this->private_->vbo_batches_[ i ]->vertex_buffer_->enable_arrays();
      this->private_->vbo_batches_[ i ]->normal_buffer_->enable_arrays();
      if ( has_values && use_colormap )
      {
        this->private_->vbo_batches_[ i ]->value_buffer_->enable_arrays();
      }
      this->private_->vbo_batches_[ i ]->faces_buffer_->draw_elements(
GL_TRIANGLES,
        static_cast< GLsizei >( ( this->private_->part_faces_[ i ].second -
        this->private_->part_faces_[ i ].first ) ), GL_UNSIGNED_INT );
      this->private_->vbo_batches_[ i ]->vertex_buffer_->disable_arrays();
      this->private_->vbo_batches_[ i ]->normal_buffer_->disable_arrays();
      if ( has_values && use_colormap )
      {
        this->private_->vbo_batches_[ i ]->value_buffer_->disable_arrays();
      }
    }
    return;
  }

  RenderResources::lock_type rr_lock( RenderResources::GetMutex() );

  // The isosurface couldn't fit in GPU memory, render it piece by piece.
  VertexAttribArrayBufferHandle vertex_buffer( new VertexAttribArrayBuffer );
  vertex_buffer->set_array( VertexAttribArrayType::VERTEX_E, 3, GL_FLOAT, 0,
0 );
  VertexAttribArrayBufferHandle normal_buffer( new VertexAttribArrayBuffer );
  normal_buffer->set_array( VertexAttribArrayType::NORMAL_E, GL_FLOAT, 0, 0
);
  VertexAttribArrayBufferHandle value_buffer;
  if ( has_values && use_colormap )
```

```
  {
    value_buffer.reset( new VertexAttribArrayBuffer );
    value_buffer->set_generic_array( 1, 1, GL_FLOAT, GL_FALSE, 0, 0 );
  }
  ElementArrayBufferHandle face_buffer( new ElementArrayBuffer );
  for ( size_t i = 0; i < num_batches; ++i )
  {
    unsigned int num_pts = this->private_->part_points_[ i ].second - this->private_-
>part_points_[ i ].first;
    ptrdiff_t vertex_size = num_pts * sizeof( PointF );
    ptrdiff_t normal_size = num_pts * sizeof( VectorF );
    ptrdiff_t value_size = has_values && use_colormap ? num_pts * sizeof( float ) :
0;
    unsigned int num_face_indices = this->private_->part_faces_[ i ].second - this-
>private_->part_faces_[ i ].first;
    ptrdiff_t face_size = num_face_indices * sizeof( unsigned int );

    vertex_buffer->set_buffer_data( vertex_size, 0, GL_STREAM_DRAW );
    void* buffer = vertex_buffer->map_buffer( GL_WRITE_ONLY );
    if ( buffer == 0 )
    {
      CORE_LOG_ERROR( "Failed to map OpenGL buffer, isosurface rendering
will"
        " be incomplete!" );
      return;
    }
    memcpy( buffer, &this->private_->points_[ this->private_->part_points_[ i ].first ],
vertex_size );
    vertex_buffer->unmap_buffer();

    normal_buffer->set_buffer_data( normal_size, 0, GL_STREAM_DRAW );
    buffer = normal_buffer->map_buffer( GL_WRITE_ONLY );
    if ( buffer == 0 )
    {
      CORE_LOG_ERROR( "Failed to map OpenGL buffer, isosurface rendering
will"
        " be incomplete!" );
      return;
    }
    memcpy( buffer, &this->private_->normals_[ this->private_->part_points_[ i
].first ], normal_size );
    normal_buffer->unmap_buffer();

    if ( has_values && use_colormap )
    {
      value_buffer->set_buffer_data( value_size, 0, GL_STREAM_DRAW );
      buffer = value_buffer->map_buffer( GL_WRITE_ONLY );
```

```
      if ( buffer == 0 )
      {
        CORE_LOG_ERROR( "Failed to map OpenGL buffer, isosurface rendering
will"
          " be incomplete!" );
        return;
      }
      memcpy( buffer, &this->private_->values_[ this->private_->part_points_[ i ].first
], value_size );
      value_buffer->unmap_buffer();
    }

    face_buffer->set_buffer_data( face_size, 0, GL_STREAM_DRAW );
    buffer = face_buffer->map_buffer( GL_WRITE_ONLY );
    if ( buffer == 0 )
    {
      CORE_LOG_ERROR( "Failed to map OpenGL buffer, isosurface rendering
will"
        " be incomplete!" );
      return;
    }
    memcpy( buffer, &this->private_->part_indices_[ i ][ 0 ], face_size );
    face_buffer->unmap_buffer();

    vertex_buffer->enable_arrays();
    normal_buffer->enable_arrays();
    if ( has_values && use_colormap )
    {
      value_buffer->enable_arrays();
    }
    face_buffer->draw_elements( GL_TRIANGLES,
      static_cast< GLsizei >( ( this->private_->part_faces_[ i ].second -
      this->private_->part_faces_[ i ].first ) ), GL_UNSIGNED_INT );
    vertex_buffer->disable_arrays();
    normal_buffer->disable_arrays();
    if ( has_values && use_colormap )
    {
      value_buffer->disable_arrays();
    }
  }
}

float Isosurface::surface_area() const
{
  lock_type lock( this->get_mutex() );
  return this->private_->area_;
}
```

```
//////////////////////////////////////////////////////////////////////////
//
// Node registry
//
// Registers/Deregisters HDX_volumeToIsosurface node
//
//////////////////////////////////////////////////////////////////////////

MStatus initializePlugin( MObject obj )
{
        MFnPlugin plugin( obj, PLUGIN_COMPANY, "1.0", "Any");
        MStatus stat = plugin.registerShape( "HDX_volumeToIsosurface",
HDX_volumeToIsosurface::id,

&HDX_volumeToIsosurface::creator,

&HDX_volumeToIsosurface::initialize,

&HDX_volumeToIsosurface::creator  );
        if ( ! stat ) {
                cerr << "Failed to register node\n";
        }

        return stat;
}

MStatus uninitializePlugin( MObject obj)
{
        MFnPlugin plugin( obj );
        MStatus stat;

        stat = plugin.deregisterNode( HDX_volumeToIsosurface::id );
        if ( ! stat ) {
                cerr << "Failed to deregister node : HDX_volumeToIsosurface \n";
        }

        return stat;
}



//////////////////////////////////////////////////////////////////////////
//
// HDX_isosurfaceToMesh (Maya node)
//
```

```
// generates the mesh polygon shape shape from the isosurface.
// this node allows to optionally display the mesh shape in the maya
// viewport (hardware rendering)
//
////////////////////////////////////////////////////////////////////////


// Maya includes
#include <math.h>
#include <maya/MIOStream.h>
#include <maya/MFnMesh.h>
#include <maya/MPoint.h>
#include <maya/MFloatPoint.h>
#include <maya/MFloatPointArray.h>
#include <maya/MIntArray.h>
#include <maya/MDoubleArray.h>
#include <maya/MFnUnitAttribute.h>
#include <maya/MFnTypedAttribute.h>
#include <maya/MFnPlugin.h>

#include <maya/MPxNode.h>
#include <maya/MObject.h>
#include <maya/MPlug.h>
#include <maya/MDataBlock.h>
#include <maya/MFnMeshData.h>

// Core includes
#include <Core/DataBlock/StdDataBlock.h>
#include <Core/Utils/StackVector.h>
#include <Core/Utils/Parallel.h>
#include <Core/Utils/Log.h>


////////////////////////////////////////////////////////////////////////
// HDX_isosurfaceToMesh
////////////////////////////////////////////////////////////////////////

class HDX_isosurfaceToMesh : public MPxNode
{
public:
                                    HDX_isosurfaceToMesh() {};
        virtual         ~HDX_isosurfaceToMesh() {};
        virtual MStatus compute(const MPlug& plug, MDataBlock& data);
        static  void*   creator();
        static  MStatus initialize();

        static MTypeId        id;
```

```
protected:
        MObject createMesh(const PointFVector&, const UIntVector& );
};

//////////////////////////////////////////////////////////////////////////

boost::shared_array<float>
computeFaceNormal(const PointF& p1, const PointF& p2, const PointF& p3)
{
  // compute face normal:
  //   U = p2 - p1
  //   V = p3 - p1
  //   Ni = UyVz - UzVy
  //   Nj = UzVx - UxVz
  //   Nk = UxVy - UyVx

  VectorF U = p2 - p1;
  VectorF V = p3 - p1;

  boost::shared_array<float> normal(new float[3]);
  normal[0] = U.y() * V.z() - U.z() * V.y();
  normal[1] = U.z() * V.x() - U.x() * V.z();
  normal[2] = U.x() * V.y() - U.y() * V.x();
  return normal;
}

//////////////////////////////////////////////////////////////////////////

MObject HDX_isosurfaceToMesh::createMesh( const PointFVector& points,
                                          const UIntVector& faces
                                        )
{

  for ( size_t i = 0; i + 2 < faces.size(); i += 3 )
  {
    size_t vertex_index1 = faces[ i ];
    size_t vertex_index2 = faces[ i + 1 ];
    size_t vertex_index3 = faces[ i + 2 ];

    // Get vertices of face
    PointF p1 = points[ vertex_index1 ];
    PointF p2 = points[ vertex_index2 ];
    PointF p3 = points[ vertex_index3 ];

    boost::shared_array<float> normal = computeFaceNormal(p1, p2, p3);
```

```
    boost::shared_array<float> vertex1(new float[POINT_LEN]);
    vertex1[0] = p1.x();
    vertex1[1] = p1.y();
    vertex1[2] = p1.z();

    boost::shared_array<float> vertex2(new float[POINT_LEN]);
    vertex2[0] = p2.x();
    vertex2[1] = p2.y();
    vertex2[2] = p2.z();

    boost::shared_array<float> vertex3(new float[POINT_LEN]);
    vertex3[0] = p3.x();
    vertex3[1] = p3.y();
    vertex3[2] = p3.z();
  }

        MObject newMesh = meshFS.create(numVertices, faces.size(),
                                                        points.size(),
faceCounts, faceConnects,
                                                        outData, &stat);

        return newMesh;
}

//////////////////////////////////////////////////////////////////////

MStatus HDX_isosurfaceToMesh::compute(const MPlug& plug, MDataBlock&
data)

{
        MStatus returnStatus;

        if (plug == outputMesh) {

                /* Get output object */

                MDataHandle outputHandle = data.outputValue(outputMesh,
&returnStatus);

                MFnMeshData dataCreator;
                MObject newOutputData = dataCreator.create(&returnStatus);

                createMesh( newOutputData, returnStatus);

                outputHandle.set(newOutputData);
                data.setClean( plug );
        } else
```

```
                return MS::kUnknownParameter;

        return MS::kSuccess;
}

//////////////////////////////////////////////////////////////////////
//
// Node registry
//
// Registers/Deregisters HDX_isosurfaceToMesh node
//
//////////////////////////////////////////////////////////////////////

MStatus initializePlugin( MObject obj )
{
        MFnPlugin plugin( obj, PLUGIN_COMPANY, "1.0", "Any");
        MStatus stat = plugin.registerShape( "HDX_isosurfaceToMesh",
HDX_isosurfaceToMesh::id,

&HDX_isosurfaceToMesh::creator,

&HDX_isosurfaceToMesh::initialize,

&HDX_isosurfaceToMesh::creator  );
        if ( ! stat ) {
                cerr << "Failed to register node\n";
        }

        return stat;
}

MStatus uninitializePlugin( MObject obj)
{
        MFnPlugin plugin( obj );
        MStatus stat;

        stat = plugin.deregisterNode( HDX_isosurfaceToMesh::id );
        if ( ! stat ) {
                cerr << "Failed to deregister node : HDX_isosurfaceToMesh \n";
        }

        return stat;
}
```

CLAIMS

What is claimed is:

1. A computer implemented method for enhanced imaging, the method comprising:

(a) transforming a non-color, non-photorealistic image into a high definition colorized photorealistic image;

(b) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

2. A computer implemented method for enhanced imaging, the method comprising:

(a) transforming a non-color, two-dimensional image generated from a diagnostic imaging device into high definition colorized, photorealistic image;

(b) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

3. The method of claim 1 or claim 2, further comprising creating animated simulations based on a plurality of said photorealistic images.

4. The method of claim 1 or claim 2, furhter comprising highlighting an area of interest in the photorealistic image using a molecular contrast promoter.

5. The method of claim 4, further comprising automatically generating a diagnosis by evaluating characteristcs of the areas of interest in the photorealistic image.

6. The method of claim 1 or claim 2, further comprising:

using functional genomics and molecular imaging to generate a molecular

contrast;

using the molecular contrast to highlight the area of interest.

7. The method of claim 6, further comprising , further comprising automatically generating a diagnosis by evaluating characteristcs of the areas of interest in the photorealistic image.

8. A computer implemented method for creating an animated simulation, the method comprising:

(a) transforming a plurality of images of a biological component, feature, characteristic, assembly, or structure, or a combination thereof, into high definition colorized photorealistic images; and

(b) assembling said photorealistic images into an animated simulation;

(c) wherein said method is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

9. An apparatus for enhanced imaging, the apparatus comprising:

(a) a computer processor; and

(b) programming in a non-transitory computer readable medium and executable on the computer processor for transforming a non-color, non-photorealistic image into a high definition colorized photorealistic image.

10. An apparatus for enhanced imaging, the apparatus comprising:

(a) a computer processor; and

(b) programming in a non-transitory computer readable medium and executable on the computer processor for transforming a non-color, two-dimensional image generated from a diagnostic imaging device into high definition colorized, photorealistic image.

11. The apparatus of claim 9 or claim 10, wherein said programming is configured to create animated simulations based on a plurality of said

photorealistic images.

12. The apparatus of claim 9 or claim 10, wherein said programming is configured to highlight an area of interest in the photorealistic image using a molecular contrast promoter.

13. The apparatus of claim 12, wherein said programming is configured for automatically generating a diagnosis by evaluating characteristcs of the areas of interest in the photorealistic image.

14. The apparatus of claim 9 or claim 10, wherein said programming is configured for performing steps comprising:

using functional genomics and molecular imaging to generate a molecular contrast; and

using the molecular contrast to highlight the area of interest.

15. The apparatus of claim 14, wherein said programming is configured for automatically generating a diagnosis by evaluating characteristcs of the areas of interest in the photorealistic image.

16. An apparatus for creating an animated simulation, the apparatus comprising:

(a) a computer processor; and

(b) programming in a non-transitory computer readable medium and executable on the computer processor for:

(i) transforming a plurality of images of a biological component, feature, characteristic, assembly, or structure, or a combination thereof, into high definition colorized photorealistic images; and

(ii) assembling said photorealistic images into an animated simulation.

17. An enhanced image, comprising:

(a) a high definition colorized photorealistic image transformed from a non-color, non-photorealistic image;

(b) wherein image transformation is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

18. An enhanced image, comprising:

(a) a high definition colorized photorealistic image transformed from a non-color, two-dimensional image generated from a diagnostic imaging device;

(b) wherein image transformation is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

19. An animated simulation, comprising:

(a) an assembly of a high definition colorized photorealistic images transformed from non-color, non-photorealistic images;

(b) wherein image transformation and assembly is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

20. An animated simulation, comprising:

(a) an assembly of a high definition colorized photorealistic images transformed from a non-color, two-dimensional image generated from a diagnostic imaging device;

(b) wherein image transformation and assembly is performed by executing programming on at least one computer processor, said programming residing on a non-transitory medium readable by the computer processor.

21. An apparatus for enhanced imaging, the apparatus comprising:

(a) a computer processor; and

(b) a non-transitory computer-readable memory storing instructions

executable by the computer processor;

(c) wherein said instructions, when executed by the computer processor, perform steps comprising:

(i) generating a database of parametric anatomy comprising one or more of volume data and isometric surface models of one or more aspects of the anatomy;

(ii) tagging one or more objects within the parametric anatomy;

(iii) inputting patient data comprising an imaging scan of a target patient anatomy of a patient;

(iv) configuring a base parametric model of patient anatomy as a function of input patient data comprising one or more physical characteristics of the patient;

(v) applying data relating to the imaging scan to the base parametric model;

(vi) searching the data relating to the imaging scan for one or more markers within the data;

(vii) aligning the parametric model to the one or more markers of the imaging scan; and

(viii) rendering the aligned parametrical model and imaging scan for photo-realistic display of the patient target anatomy.

22. The apparatus of claim 21, wherein data relating to the imaging scan comprises DICOM data from one or more of an MRI, CT, or ultrasound scan of the patient.

23. The apparatus of claim 21, wherein the database is generated by acquiring input from patient data comprising one or more of patient scans, statistics or photos relating to patient.

24. The apparatus of claim 21, wherein the isometric surface models are configured for photo-real real-time VR rendering.

25. The apparatus of claim 21, wherein the tagged data is configured so that that can be turned on or off for viewing.

26. The apparatus of claim 21, the instructions further comprising:
allowing manual input for selecting the one or more markers.

27. The apparatus of claim 22, wherein aligning the parametric model to the one or more markers of the imaging scan comprises isometric surface alignment of the parametric anatomical geometry to the patient's DICOM scan data.

28. The apparatus of claim 27, wherein the alignment is done taking into account input patient data relating to one or more of weight, height, BMI, X-ray, and patient photos.

29. The apparatus of claim 27, wherein the alignment is performed on both volume data and isometric surfaces.

30. The apparatus of claim 29, wherein the alignment is adjusted by manual input via selecting one or more structures on slices of the MRI or CT scan to fine-tune the base parametric model.

31. The apparatus of claim 21, the instructions further configured for:
applying photographic reference of skin color to the output parametric model.

32. The apparatus of claim 21, the instructions further configured for:
auto-alignment and projection of one or more of the following to the parametric model: patient wounds, surgeon markups, X-rays, notes and other text files.

33. A computer implemented method for enhanced imaging, the method comprising:

generating a database of parametric anatomy comprising one or more of volume data and isometric surface models of one or more aspects of the anatomy;

tagging one or more objects within the parametric anatomy;

inputting patient data comprising an imaging scan of a target patient anatomy of a patient;

configuring a base parametric model of patient anatomy as a function of input patient data comprising one or more physical characteristics of the patient;

applying data relating to the imaging scan to the base parametric model;

searching the data relating to the imaging scan for one or more markers within the data;

aligning the parametric model to the one or more markers of the imaging scan; and

rendering the aligned parametrical model and imaging scan for photo-realistic display of the patient target anatomy.

34. The method of claim 33, wherein data relating to the imaging scan comprises DICOM data from one or more of an MRI, CT, or ultrasound scan of the patient.

35. The method of claim 33, wherein the database is generated by acquiring input from patient data comprising one or more of patient scans, statistics or photos relating to patient.

36. The method of claim 33, wherein the isometric surface models are configured for photo-real real-time VR rendering.

37. The method of claim 33, wherein the tagged data is configured so that that can be turned on or off for viewing.

38. The method of claim 33, the method further comprising:
allowing manual input for selecting the one or more markers.

39. The method of claim 34, wherein aligning the parametric model to the one or more markers of the imaging scan comprises isometric surface alignment of the parametric anatomical geometry to the patient's DICOM scan data.

40. The method of claim 39, wherein the alignment is done taking into account input patient data relating to one or more of weight, height, BMI, X-ray, and patient photos.

41. The method of claim 39, wherein the alignment is performed on both volume data and isometric surfaces.

42. The method of claim 41, wherein the alignment is adjusted by manual input via selecting one or more structures on slices of the MRI or CT scan to fine-tune the base parametric model.

43. The method of claim 33, the method further comprising:
applying photographic reference of skin color to the output parametric model.

44. The method of claim 33, the method further comprising:
auto-alignment and projection of one or more of the following to the parametric model: patient wounds, surgeon markups, X-rays, notes and other text files.

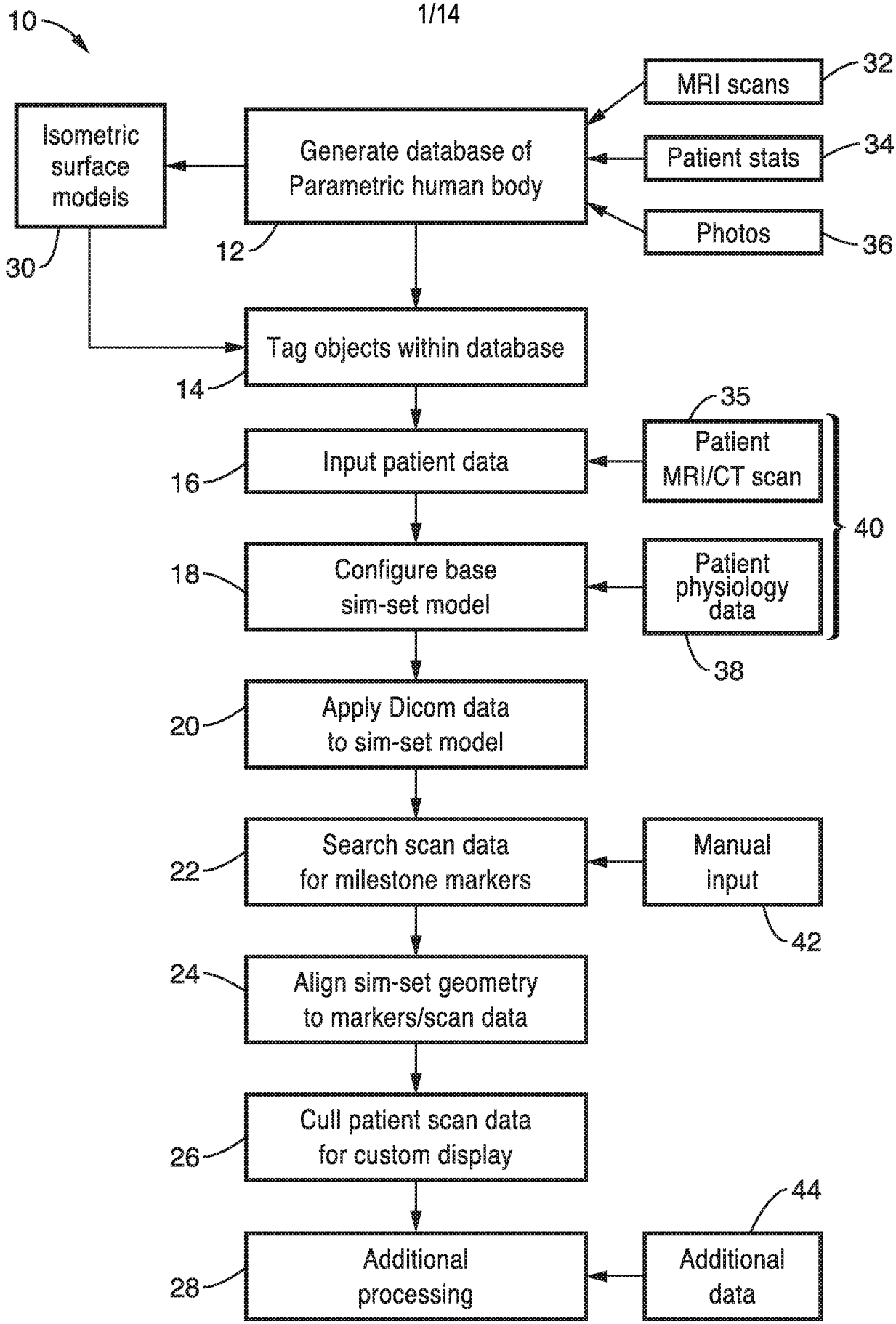

10
MRI scans
32
Isometric surface models
Generate database of Parametric human body
Patient stats
34
Photos
36
30
12
Tag objects within database
14
35
Patient MRI/CT scan
Input patient data
16
40
Configure base sim-set model
18
Patient physiology data
38
Apply Dicom data to sim-set model
20
Search scan data for milestone markers
22
Manual input
42
Align sim-set geometry to markers/scan data
24
Cull patient scan data for custom display
26
44
Additional data
Additional processing
28

FIG. 1

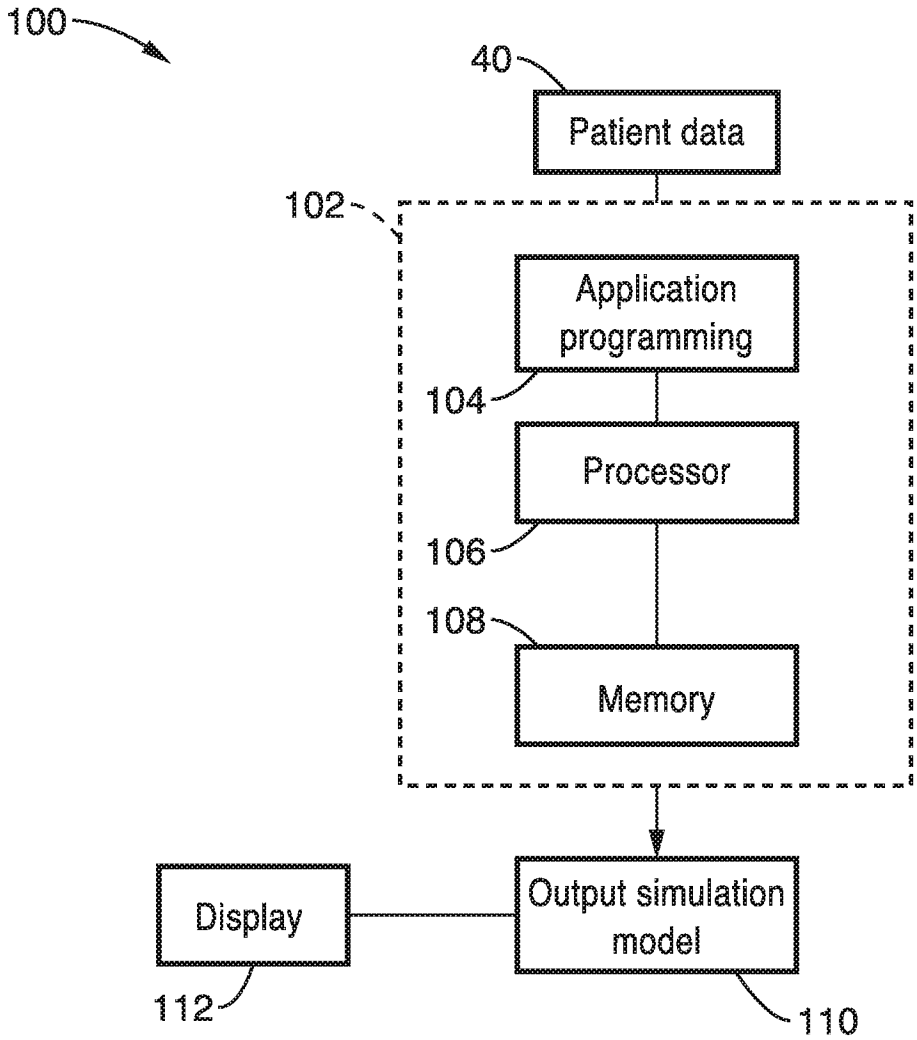

100
40
Patient data
102
Application programming
104
Processor
106
108
Memory
Display
Output simulation model
112
110

FIG. 2

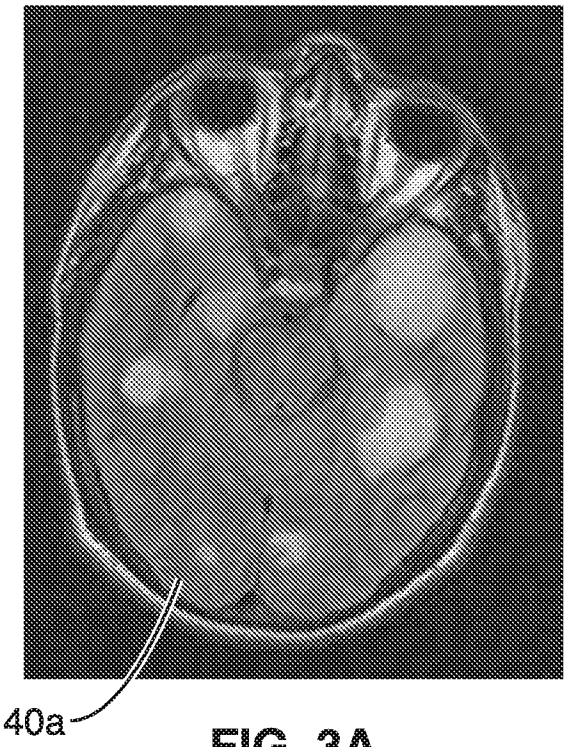


FIG. 3A

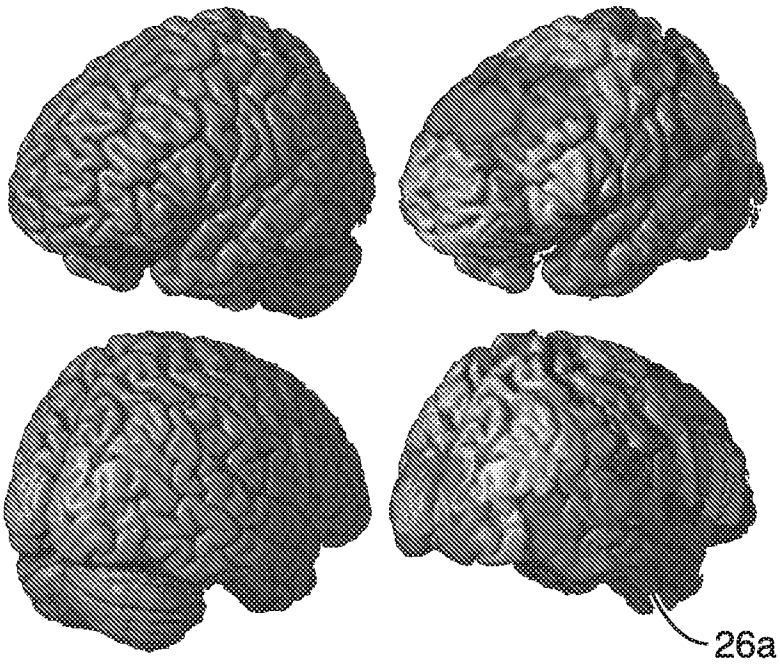


FIG. 3B

FIG. 4A

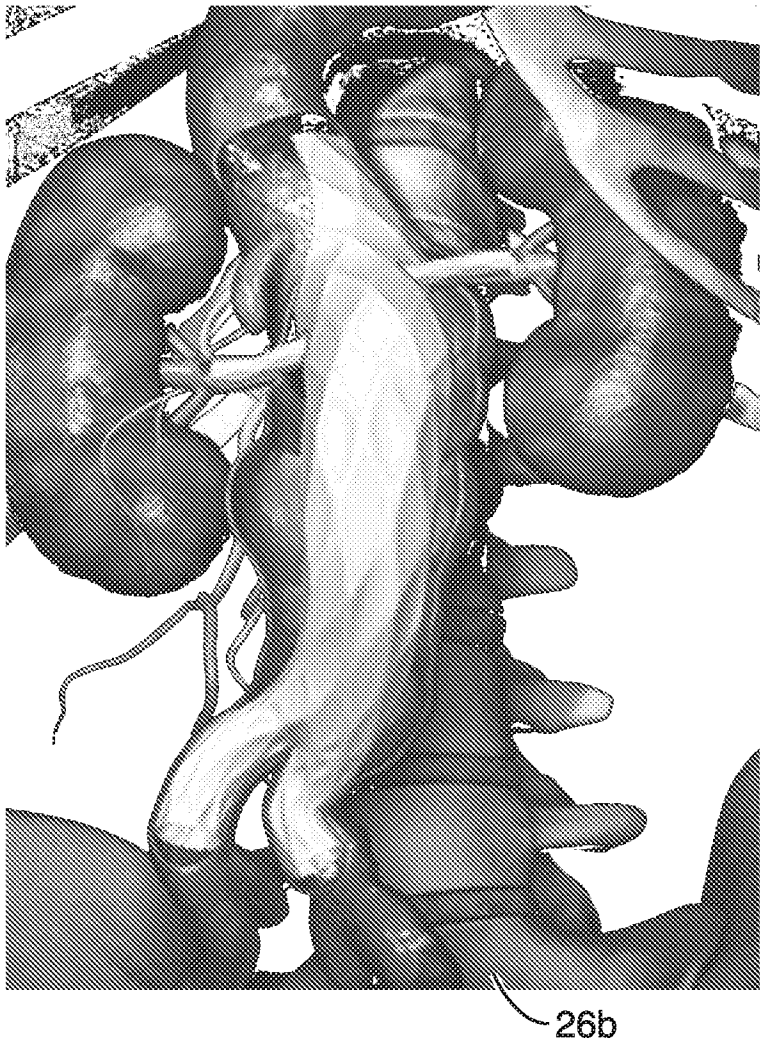


FIG. 4B

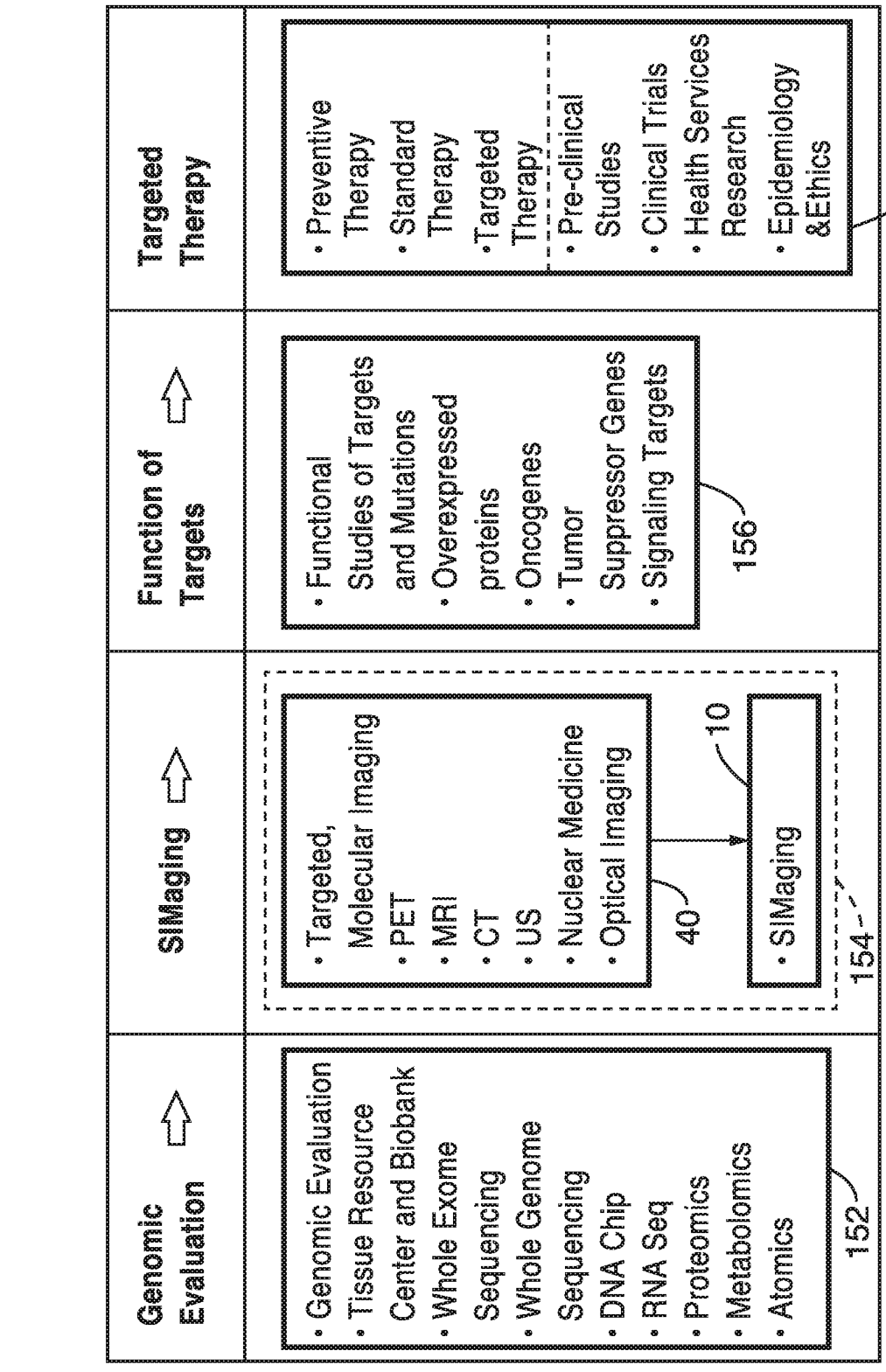

150
Genomic Evaluation
SiMaging
Function of Targets
Targeted Therapy
• Genomic Evaluation
• Tissue Resource Center and Biobank
• Whole Exome Sequencing
• Whole Genome Sequencing
• DNA Chip
• RNA Seq
• Proteomics
• Metabolomics
• Atomics
152
• Targeted, Molecular Imaging
• PET
• MRI
• CT
• US
• Nuclear Medicine
• Optical Imaging
40
10
• SIMaging
154
• Functional Studies of Targets and Mutations
• Overexpressed proteins
• Oncogenes
• Tumor Suppressor Genes
• Signaling Targets
156
• Preventive Therapy
• Standard Therapy
• Targeted Therapy
• Pre-clinical Studies
• Clinical Trials
• Health Services Research
• Epidemiology &Ethics
158

FIG. 5

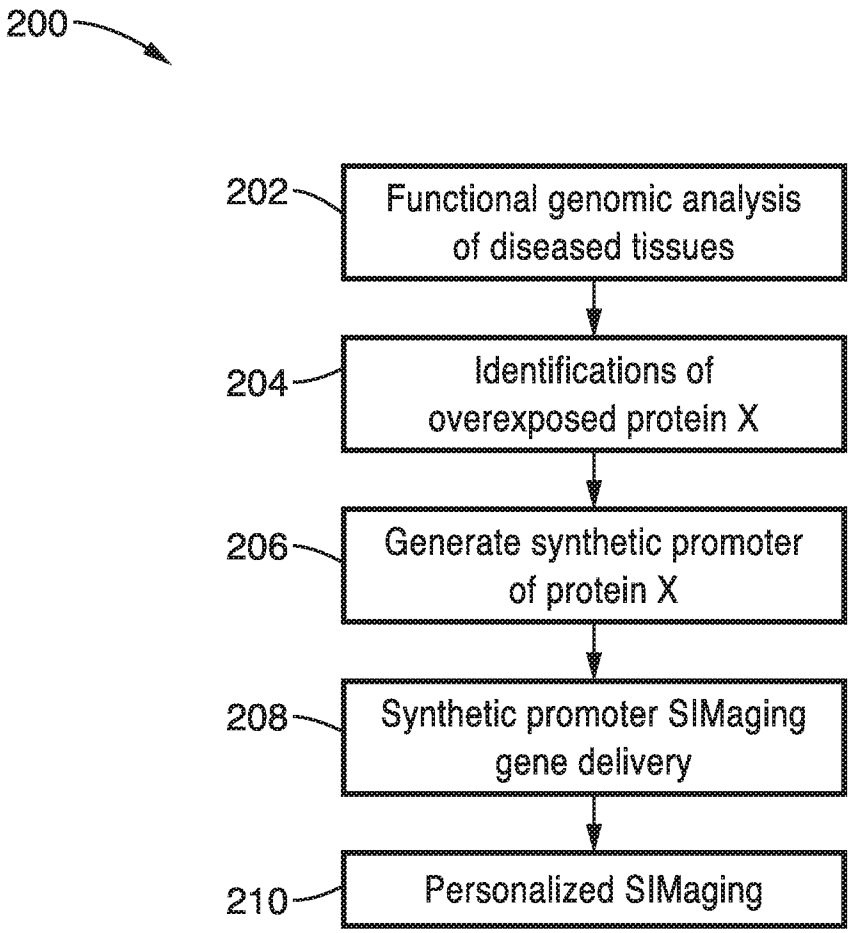

200
202
Functional genomic analysis of diseased tissues
204
Identifications of overexposed protein X
206
Generate synthetic promoter of protein X
208
Synthetic promoter SIMaging gene delivery
210
Personalized SIMaging

FIG. 6

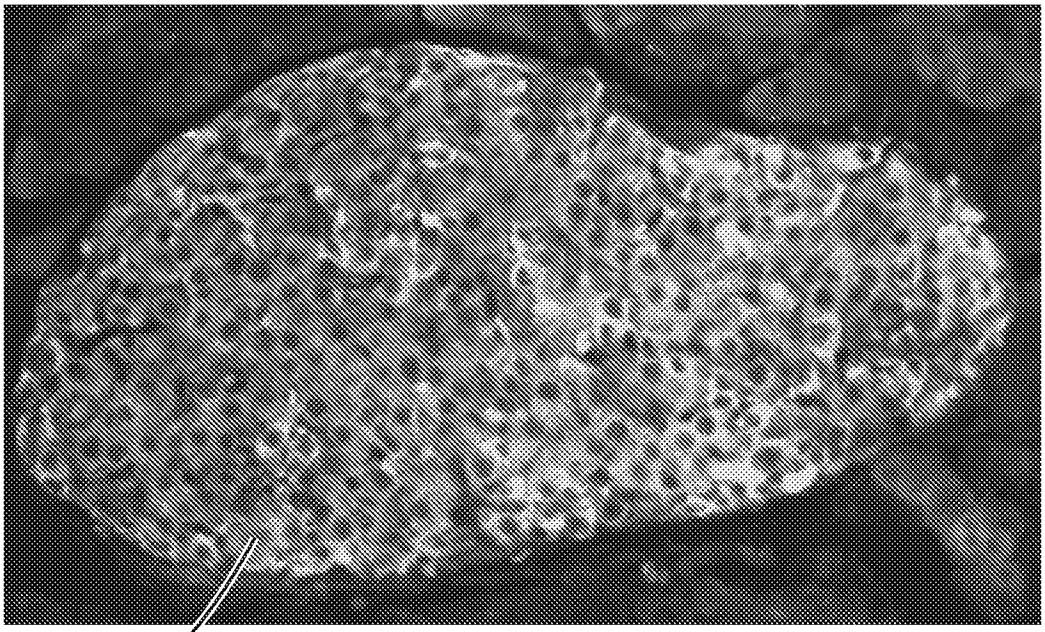

40c

FIG. 7A

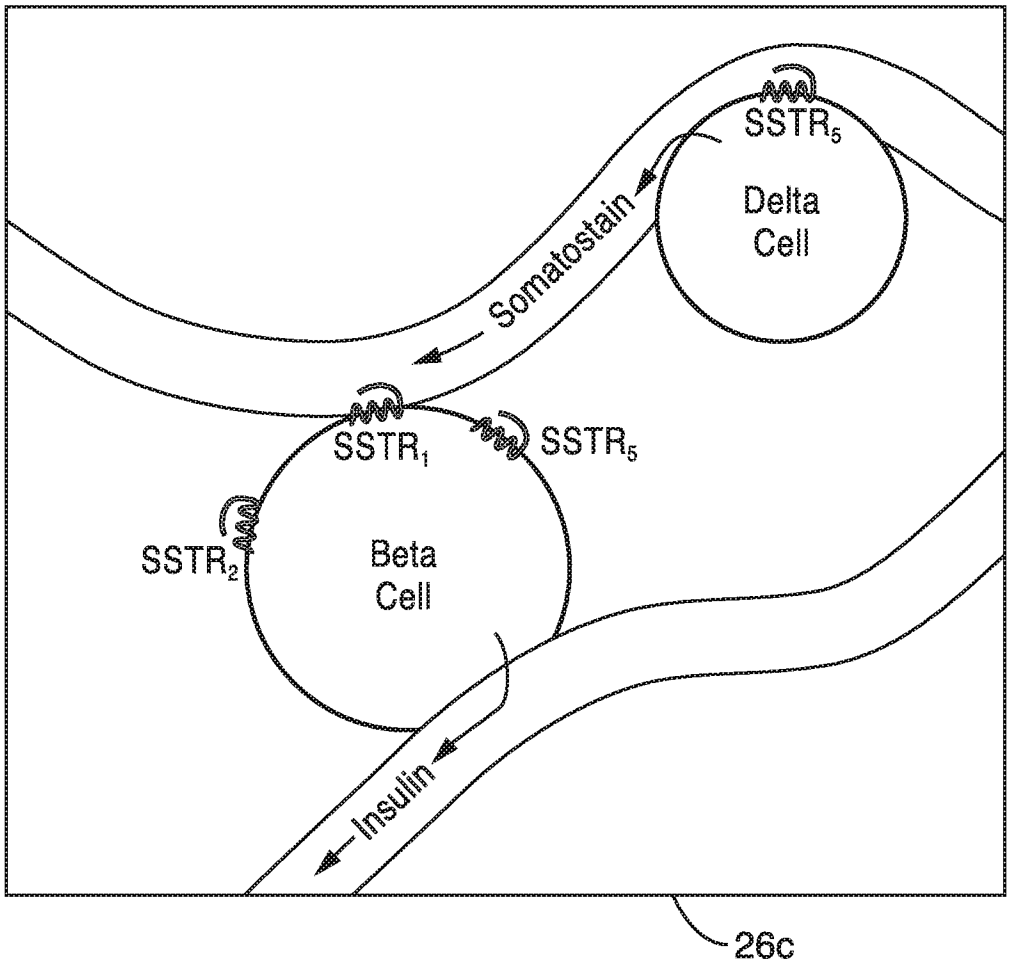

SSTR5
Delta Cell
Somatostain
SSTR1
SSTR5
SSTR2
Beta Cell
Insulin
26c

FIG. 7B

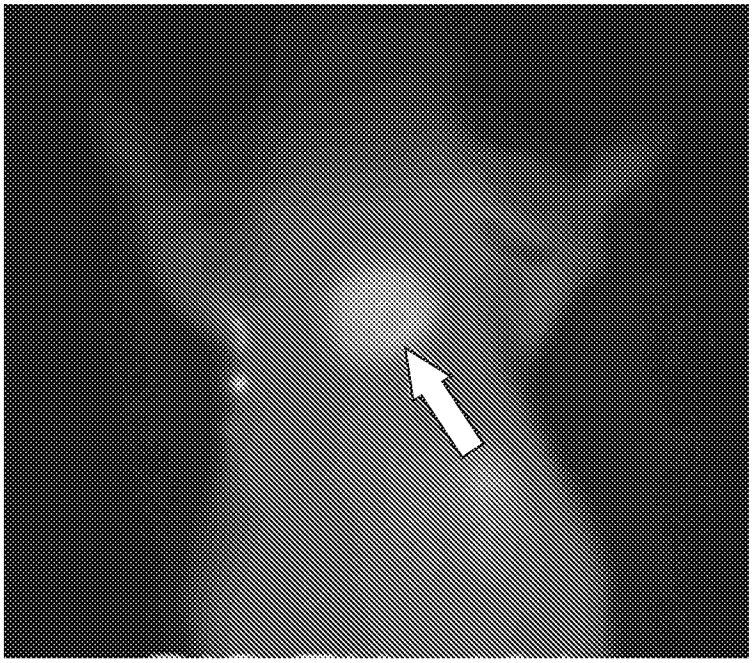


FIG. 8A

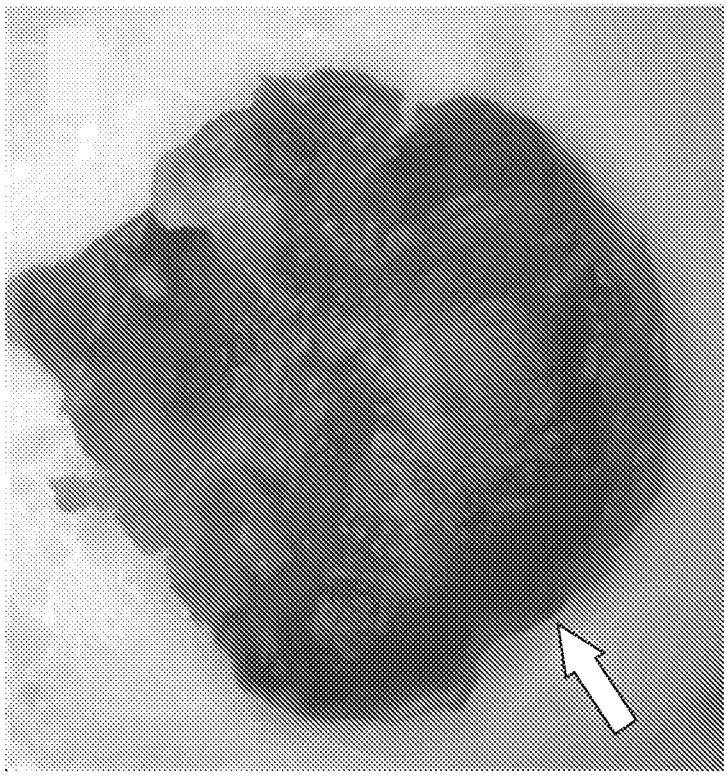
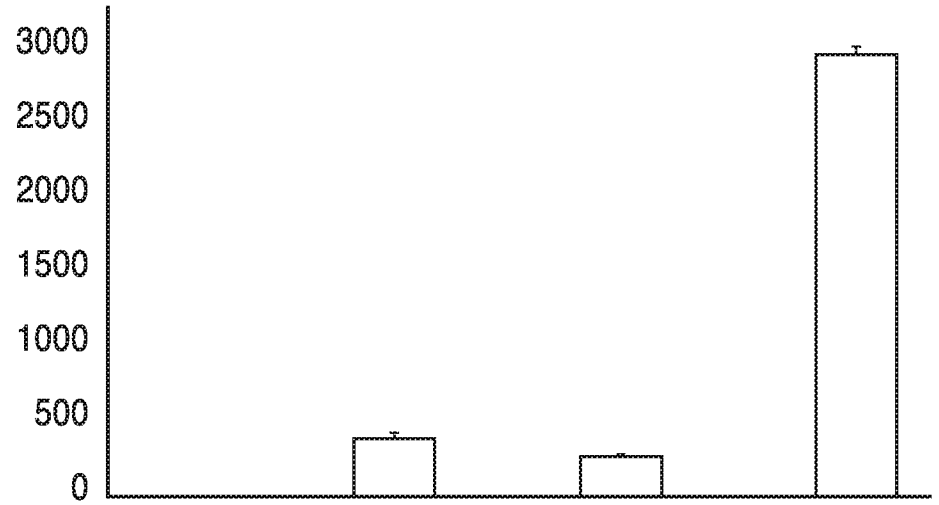


FIG. 8B



	pFOX-CAT	RIP-CAT	HIP-CAT	SHIP-CAT
PANC-1	0	370	246	2897
HPDE	0	0	14	5

FIG. 9A

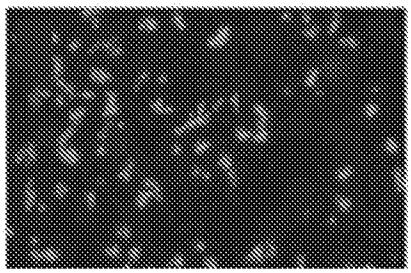


FIG. 9B

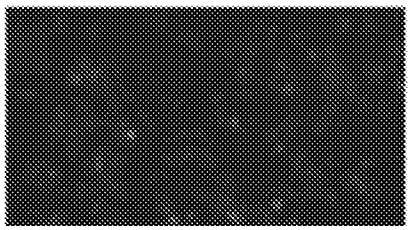


FIG. 9C

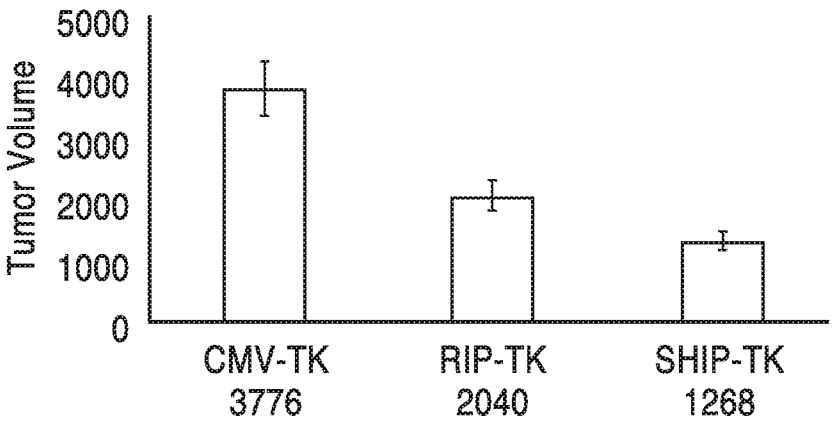


FIG. 9D

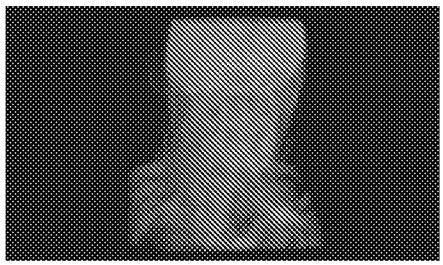


FIG. 10A

FIG. 10B

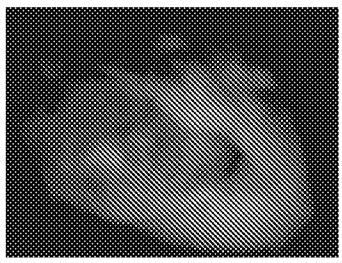


FIG. 10C

FIG. 10D

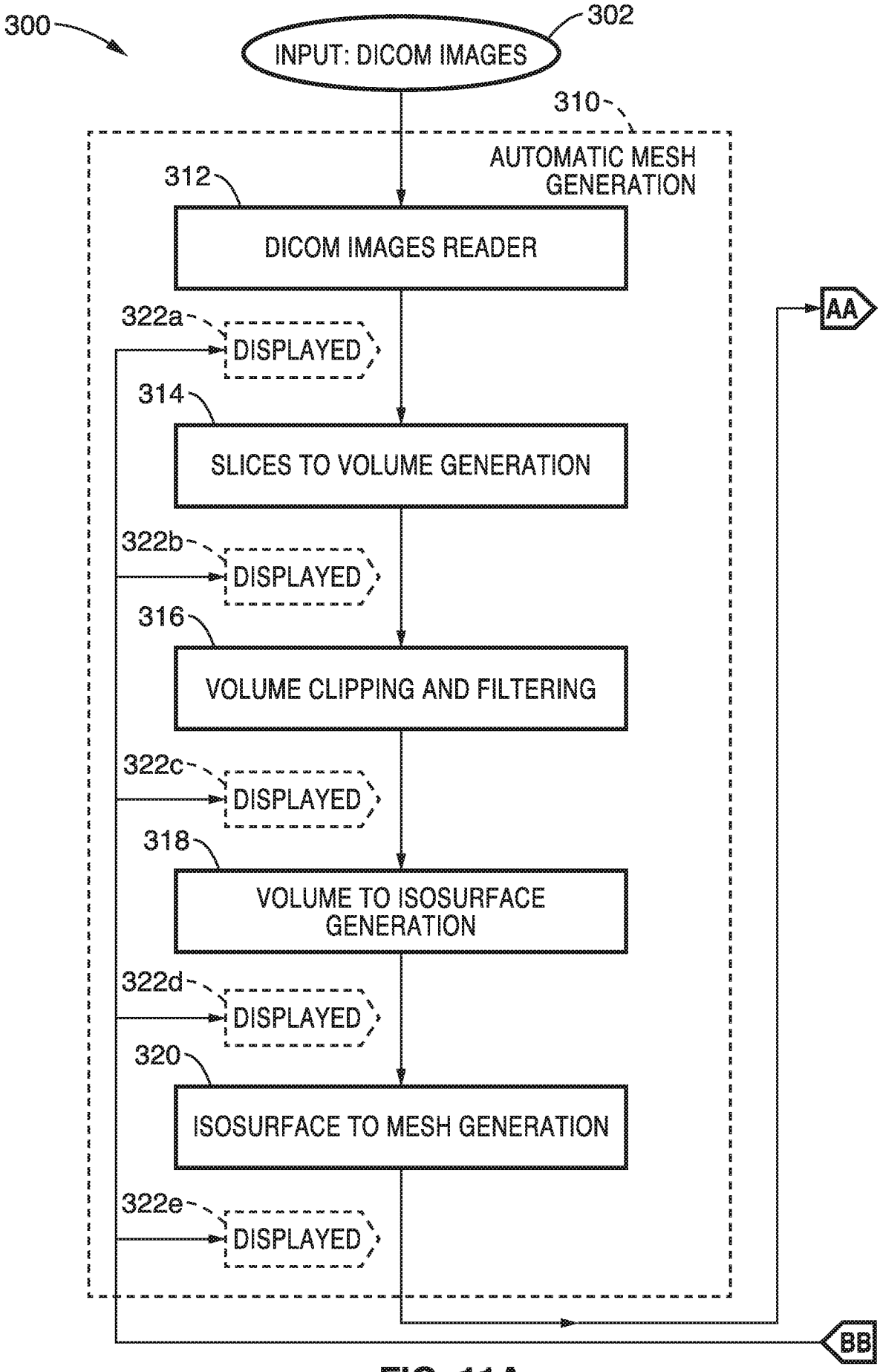

300
302
INPUT: DICOM IMAGES
310
AUTOMATIC MESH GENERATION
312
DICOM IMAGES READER
322a
DISPLAYED
314
SLICES TO VOLUME GENERATION
322b
DISPLAYED
316
VOLUME CLIPPING AND FILTERING
322c
DISPLAYED
318
VOLUME TO ISOSURFACE GENERATION
322d
DISPLAYED
320
ISOSURFACE TO MESH GENERATION
322e
DISPLAYED
AA
BB

FIG. 11A

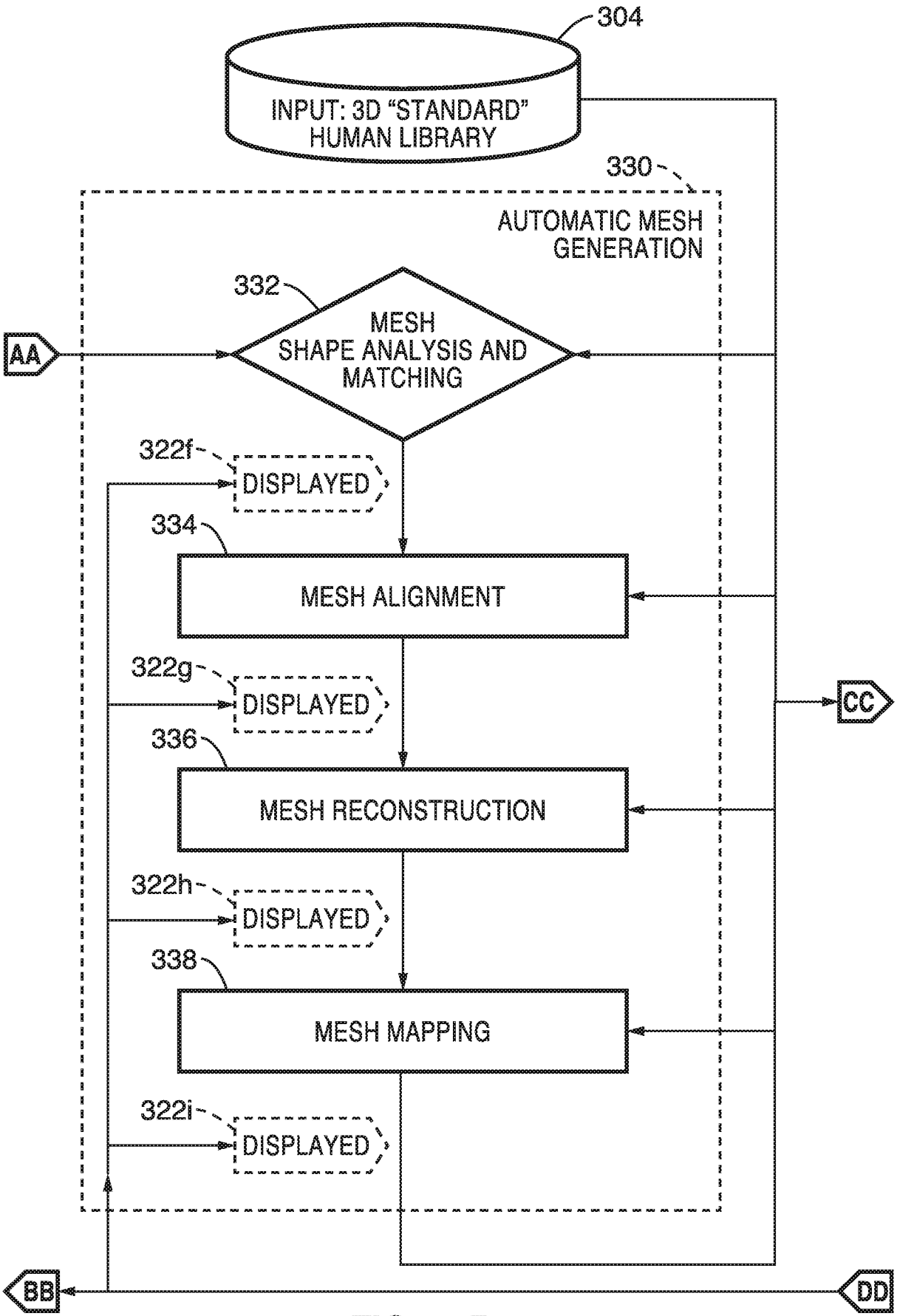

304
INPUT: 3D "STANDARD" HUMAN LIBRARY
330
AUTOMATIC MESH GENERATION
332
MESH SHAPE ANALYSIS AND MATCHING
AA
322f
DISPLAYED
334
MESH ALIGNMENT
322g
DISPLAYED
CC
336
MESH RECONSTRUCTION
322h
DISPLAYED
338
MESH MAPPING
322i
DISPLAYED
BB
DD

FIG. 11B

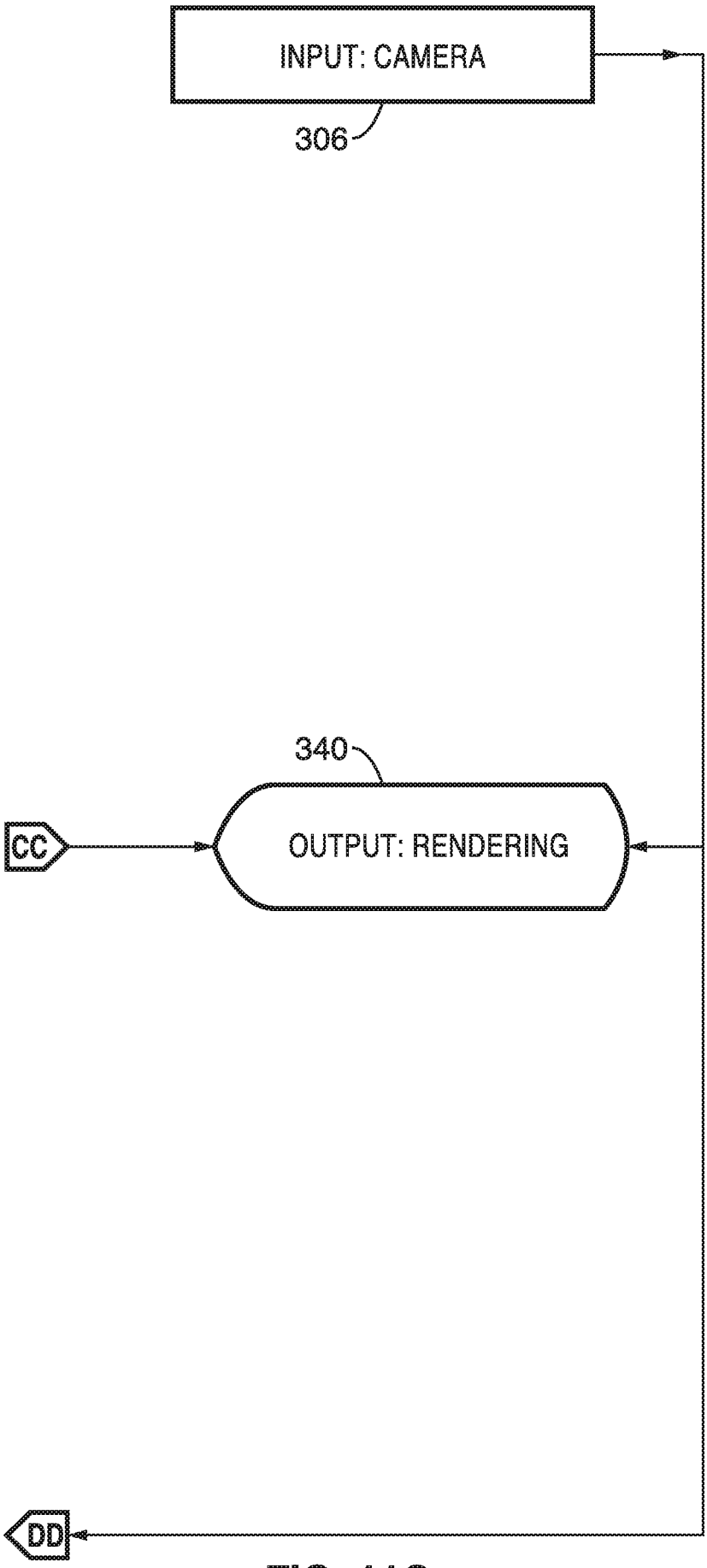


FIG. 11C

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2017/023669

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - A61B 34/10; A61B 5/00; A61B 19/00; G06K 9/34 (2017.01)
CPC - A61B 34/10; G06F 19/3437; G06T 3/0012; G06T 15/02 (2017.02)

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC - 382/173; 382/190; 700/98 (keyword delimited)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2013/0278600 A1 (CHRISTENSEN et al) 24 October 2013 (24.10.2013) entire document	1, 3, 9, 12, 18, 20
X — Y	US 6,233,480 B1 (HOCHMAN et al) 15 May 2001 (15.05.2001) entire document	2, 4-7, 10, 13-16, 19 — 8, 17, 21
X	WO 2014/145267 A1 (CONFORMIS, INC.) 18 September 2014 (18.09.2014) entire document	22-45
Y	US 2010/0157018 A1 (LAMPOTANG et al) 24 June 2010 (24.06.2010) entire document	8, 17, 21
A	US 2008/0134094 A1 (SAMADANI et al) 05 June 2008 (05.06.2008) entire document	1-10, 12-45
A	WO 2014/159082 A1 (BLUE BELT TECHNOLOGIES, INC.) 02 October 2014 (02.10.2014) entire document	1-10, 12-45
A	US 2007/0070063 A1 (SUMANAWEERA) 29 March 2007 (29.03.2007) entire document	1-10, 12-45
A	US 2009/0184955 A1 (THIELE) 23 July 2009 (23.07.2009) entire document	1-10, 12-45
A	WO 2009/004296 A1 (YANG et al) 08 January 2009 (08.01.2009) entire document	1-10, 12-45
A	US 8,674,989 B1 (DALAL et al) 18 March 2014 (18.03.2014) entire document	1-10, 12-45
A	US 6,359,618 B1 (HEIRICH) 19 March 2002 (19.03.2002) entire document	1-10, 12-45

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 July 2017	21 JUL 2017
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, VA 22313-1450 Facsimile No. 571-273-8300	Authorized officer Blaine R. Copenheaver PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2017/023669

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:
See extra sheet(s).

1. ☒ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☒ No protest accompanied the payment of additional search fees.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2017/023669

Continued from Box No. III Observations where unity of invention is lacking

This application contains the following inventions or groups of inventions which are not so linked as to form a single general inventive concept under PCT Rule 13.1. In order for all inventions to be examined, the appropriate additional examination fees must be paid.

Group I, claims 1-21, drawn to creating an animated simulation.
Group II, claims 22-45, drawn to generating a database of parametric anatomy.

The inventions listed as Groups I-II do not relate to a single general inventive concept under PCT Rule 13.1 because, under PCT Rule 13.2, they lack the same or corresponding special technical features for the following reasons: the special technical feature of the Group I invention: transforming a non-color, non-photorealistic image into a high definition colorized photorealistic image as claimed therein is not present in the invention of Group II. The special technical feature of the Group II invention: generating a database of parametric anatomy comprising one or more of volume data and isometric surface models of one or more aspects of the anatomy; tagging one or more objects within the parametric anatomy; inputting patient data comprising an imaging scan of a target patient anatomy of a patient; configuring a base parametric model of patient anatomy as a function of input patient data comprising one or more physical characteristics of the patient; applying data relating to the imaging scan to the base parametric model; searching the data relating to the imaging scan for one or more markers within the data; aligning the parametric model to the one or more markers of the imaging scan; and rendering the aligned parametrical model and imaging scan for photo-realistic display of the patient target anatomy as claimed therein is not present in the invention of Group I.

Groups I and II lack unity of invention because even though the inventions of these groups require the technical feature of an apparatus for enhanced imaging of photo-realistic images, this technical feature is not a special technical feature as it does not make a contribution over the prior art.

Specifically, US 2008/0134094 A1 (SAMADANI et al) 05 June 2008 (05.06.2008) teaches an apparatus for enhanced imaging of photo-realistic images (Paras. 92-96).

Since none of the special technical features of the Group I or II inventions are found in more than one of the inventions, unity of invention is lacking.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2017/023669

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 2007/0154110 A1 (WEN et al) 05 July 2007 (05.07.2007) entire document	1-10, 12-45