PTO/SB/16 (6-95)
Approved for use through 04/11/98. OMB 0651-0037
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
(Reproduction)

11/26/97 JC561 U.S.PTO

A

PROVISIONAL APPLICATION COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION under 37 CFR 1.53(b)(2).

/PROV

Docket Number	8418/1	Type a plus sign (+) inside this box →	+

INVENTOR(s)/APPLICANT(s)

LAST NAME	FIRST NAME	MIDDLE INITIAL	RESIDENCE (CITY AND EITHER STATE OR FOREIGN COUNTRY)
JASZLICS	Ivan	J.	Golden, Colorado, U.S.A.
JASZLICS	Sheila	L.	Golden, Colorado, U.S.A.

TITLE OF THE INVENTION (280 characters max)

SYSTEM FOR COMBINING VIRTUAL IMAGES WITH REAL-WORLD SCENES

CORRESPONDENCE ADDRESS

Dorr, Carson, Sloan & Birney, P.C.
3010 E. 6th Avenue
Denver

STATE	CO	ZIP CODE	80206	COUNTRY	U.S.A.

ENCLOSED APPLICATION PARTS (*check all that apply*)

☒	Specification	Number of Pages	142	☒	Small Entity Statement
☒	Drawing(s)	Number of Sheets	19	☒	Other (specify) Assignment

METHOD OF PAYMENT (*check one*)

☒	A check or money order is enclosed to cover the Provisional filing fees	PROVISIONAL FILING FEE AMOUNT ($)	$75.00
☐	The Commissioner is hereby authorized to charge filing fees and credit Deposit Account Number 04-1414		

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☒ No.

☐ Yes, the name of the U.S. Government agency and the Government contract number are : _____

Respectfully submitted,

SIGNATURE Thomas S Birney Date 11/26/97

TYPED or PRINTED NAME Thomas S. Birney REGISTRATION NO. (*if appropriate*) 30,025

☐ Additional inventors are being named on separately numbered sheets attached hereto

PROVISIONAL APPLICATION FILING ONLY

SYSTEM FOR COMBINING VIRTUAL IMAGES WITH REAL-WORLD SCENES

BACKGROUND OF THE INVENTION

Field of the Invention. The present invention relates generally to the field of virtual reality simulation, and more particularly to systems for combining virtual images with real-world scenes.

Statement of the Problem. Virtual reality systems have gained wide acceptance in recent years for training and simulations in a number of fields, including aircraft simulators, virtual reality games, surgical training, and military training. Conventional virtual reality systems generate a field of view for the user that is either completely computer-generated, or may include real-world scenery as background. Some virtual reality systems use portions of real-world images (e.g., a particular object, pattern, or texture) that can be incorporated into a computer-generated environment. However, conventional virtual reality systems do no typically incorporate virtual images into real-world scenes due to the difficulties of integrating a virtual image into a real-world scene in a realistic manner. For example, the virtual image can mask portions of the real-world scene, and objects in the real-world scene can mask portions of the virtual image depending on their relative locations and sizes.

A need exists in many types of simulations to combine virtual images with real-world scenes. For example, the U.S. Army trains its

armored vehicle crews using virtual reality simulators. These simulators are static and rely entirely on computer-generated images. The army also trains armored vehicle crews in war games using actual vehicles moving over real terrain. These vehicles can be fully instrumented and connected to computer systems that record and analyze the performance of the crew and vehicle throughout the exercise. The computer system can also record and indicate simulated "hits" on opposing vehicles. Although such war games are much more realistic than simulators, they are also much more expensive and time consuming due to the requirements of operating actual vehicles and managing the manpower necessary to conduct a realistic exercise. The cost of instrumentation for a large number of vehicles is also substantial. The present invention can be used in these war games to generate virtual tanks that appear move about the terrain. Virtual vehicles and real vehicles can appear in the same scene. The present invention can also be used to generate virtual "hits" and explosions during an exercise.

The present invention also has application in the field of aircraft simulation. Virtual aircraft or virtual flying conditions can be combined with real-world scenes during an actual flight or a simulated flight. Another possible field of use is in games, such as laser tag, so that virtual players, objects, and special effects can be combined with real players and real objects in the playing field.

Solution to the Problem. The present system enables computer-generated virtual images to be combined with images of the real world. A range scanner determines the shape and distance of real-world objects within a field of regard of interest to the observer. Virtual masking objects, which are simplified computer models of real-

world objects, are derived from the range data. A computer simulates virtual entities and combines these virtual images with the virtual masking objects to create masked virtual images. The masked virtual images show the portions of virtual entities that would be visible if the virtual entities actually existed in the real world. The masked virtual images and the real-world scene are combined and displayed in such a manner that the virtual images appear to be obscured, when appropriate for their simulated location relative to real-world objects.

60065904.112597

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention can be more readily understood in conjunction with the accompanying drawings, in which:

Figures 1a through 1d are examples of a real-world image, a virtual image, the resulting masked virtual image, and combined images, respectively.

Figure 2 is a simplified block diagram showing the major steps in the present method.

Figure 3 is an example of a range map generated by the range scanner.

Figure 4 is an example of the virtual masking objects corresponding to the range map shown in Figure 3.

Figure 5 is a VRML listing of a virtual masking object display construct representing one of the grassy knolls shown in Figures 3 and 4.

Figure 6 is an example of the effect of implementing the virtual masking object defined in Figure 5 on the virtual image of a cube behind the knoll and the virtual image of a cone in front of the knoll.

Figure 7 is an image combining the grassy knoll, cube, and cone.

Figure 8 is a listing of the equations used to implement time extrapolation for the virtual masking objects between range frame updates.

Figure 9 is a three-dimensional example of a field of regard illustrating the use of time processing algorithms.

Figure 10 is a pseudo-code listing of an algorithm for digital image combination.

Figure 11 is a simplified diagram showing one embodiment for injecting virtual images into a real-world scene.

Figure 12 is a simplified diagram showing an alternative embodiment for combining a virtual image with a real-world image.

Figure 13 is an example of a video image combining the virtual image from Figure 6 with a real-world scene.

Figure 14 is an example of a combined view in the infrared as seen through the thermal viewer of a tank or infantry fighting vehicle.

Figure 15 is an example of a combined view depicting conceptual virtual entities.

Figure 16 is a simplified block diagram of the reference apparatus showing one possible implementation of the present system.

Figure 17 is a simplified diagram of the transmitter section of the laser range scanner.

Figure 18 is a simplified diagram of the receiver section of the laser range scanner.

Figure 19 is a diagram of the scanning pattern for the laser range scanner.

Figure 20 is a simplified block diagram of the virtual masking object (VMO) generation software.

Figure 21 is an example of the control frame in the computer-generated image.

Figure 22 is a simplified block diagram of the virtual simulation and control software.

Figure 23 is a simplified block diagram demonstrating how an input laser beam can be steered using a crystal excited by an alternating electrical field.

Figure 24 is a simplified block diagram illustrating passive optical ranging using two cameras to determine the range of an object.

DETAILED DESCRIPTION OF THE INVENTION

An example of the basic functionality of the present invention is demonstrated in Figures 1a through 1d. The present system combines the image of a real-world scene (Figure 1a) with a virtual image (Figure 1b). The invention provides a method and the description of a reference apparatus and alternative implementations, to sense and remove elements of a virtual image, thereby generating a masked virtual image (Figure 1c). The masked virtual image and the image of the real world are then combined in such a manner that in the resulting combined image (Figure 1d) the virtual object will appear to exist and move in the real world at its virtual position

The block diagram of Figure 2 shows the major steps of the present method. One implementation of the present invention is diagrammed in Figures 16 through 22.

Range Scanning. Turning to Figure 2, a range scanner 101 determines the distance of the entities present in a real-world scene 102 within small ranging sectors. Each ranging sector is defined as a small angular sector (e.g., 1 milliradian in height and width) and by its azimuth and elevation, seen from a momentary observation point 100. The ranging sectors are included within the field of regard 103 of the range scanner 101. Scanning is limited in range, for example to 1000 meters. Range scanning may be performed by a laser range scanner (as in the reference apparatus shown in Figures 16 - 22) or by other means that can determine the distance of objects within the angular space of ranging sector. Range data 104 are generated by the range scanning step, and include azimuth, elevation, and range to the nearest object for each ranging sector.

Range scanning is an existing technology that provides range data necessary for virtual masking object generation. Normally, existing ranging systems will be used. The type of scanning will depend on the range and ranging sector requirements of a particular implementation of the invention for a specific application. For a range requirement of approximately 400 meters, and angular resolution of approximately 1 milliradian, a 905 nanometer eye-safe infrared laser scanner was sufficient in the reference apparatus.

In lieu of laser range scanning, other types of range scanning can provide sufficient range data for the present system. Examples of other scanning methods include three-dimensional radar scanning for longer ranges (multiple kilometers), or acoustical ranging for short ranges (e.g., within 50 meters). Active flash ranging methods, in which the acquisition of range for the ranging sectors is achieved in a simultaneous manner, covering the whole field of regard by phase-modulated laser or radar ranging flashes, or by multiple amplitude-modulated laser or radar pulses, are also applicable for short ranges. Passive ranging methods (e.g., stereographic or parallax comparison of real-world scene elements) can also be used, when their angular resolution fits the requirements of a specific application.

The frequency of range data updates imposed on the range scanner depends on the type of real-world scene considered for specific implementations of the invention. A 266 Hz laser pulse rate was used for scanning the static scenes considered for the reference apparatus in Figures 16 - 22. The field of regard was scanned from a static observation point with approximately 26,000 ranging elements to provide a range frame update rate in the order of 0.01 Hz. For dynamic scenes with a moving observation point and mobile elements within the real-world scene, the system will normally require

considerably higher range scanning pulse rates, such as 30 KHz, and a range frame update rate for the field of regard in the order of 1 Hz or higher.

Range, azimuth, and elevation data may be provided directly for each ranging sector, or indirectly, by providing only the range in each ranging sector, the scanning row number in the case of horizontal scanning for each row, and a frame start and stop signal. In this case, the angular width and height of each ranging sector is also provided.

Virtual Masking Object Generation. Virtual masking object generation 105 is a key step of the invention. The ranging data of the ranging sectors are used to define the virtual masking objects 106 corresponding to real world objects. The range map obtained in a 0.261 radians wide and 0.096 radians high field of regard by a 905 nanometer wavelength infrared scanning laser, scanning an area of small grassy hillocks, a parked automobile van, and a tree is shown in Figure 3. The range return for each 0.98 milliradian x 0.98 milliradian ranging sector is encoded as a gray intensity value, longer distances are shown in darker shades. Areas of no return are shown in white. Such range maps are raw data, normally requiring further processing.

Rectangular virtual masking objects can be generated from the raw data at the measured range, having the angular width and height of each ranging sector, to mask portions of the images of such virtual entities as may lie beyond this virtual masking object. This is one method within the scope of the invention for virtual masking object generation. Another, alternative approach is to apply a statistical filter to the raw ranging data in order to establish coherent, larger virtual masking objects (Fig. 4). The software code used for generating

larger virtual masking objects, as shown in Figure 4, is listed in Appendix A. The conditions for establishing coherence of virtual masking objects in the software code of Appendix A are listed in Table 1. These conditions are associated with the specific method of ranging in the reference apparatus, which uses horizontal scan lines, advancing vertically from scan line to scan line.

Alternative rules may be used for establishing the virtual masking objects. An example of an alternative method is direct three-dimensional filtering. With this method, individual points defined by their ranging sector's azimuth, elevation and range are added to a virtual masking object as long as new points fall, within a pre-determined tolerance "t" on a common plane, and when such points are contiguous within the tolerance of a hole-filling algorithm.

The above methods result in a geometrical definition of the virtual masking objects. These may be simple quadrangles for a single ranging sector, or complex agglomerations of many ranging sectors into a two or three-dimensional surface. Within the virtual masking object generation step, three additional functions are performed.

Display Construct Generation. Display construct generation builds data constructs compatible with a specific rendering system. For example, the virtual masking objects shown in Figure 4 were prepared for presentation under the Virtual Reality Modeling Language (VRML), version 2.0 (International Standards Organization ISO/IEC CD 14772), using the "IndexedFaceSet" high-level virtual reality node specification. This is reflected in the last section of the software code in Appendix A written in the Java language. Other sections of the software code are generic and independent of the

actual rendering system used. The invention itself is independent of any specific rendering implementation.

Display Attribute Preparation. Display attribute preparation is the last processing function of the virtual masking object generation step. This function may be separate from, or coincident with the display construct generation function, depending on the specific rendering system and image combination system used. The virtual masking objects must be rendered with display attributes that permit the automatic replacement of their pixels in the image combination step with a video background representing the external world. This attribute depends on the specific image combination hardware, and is usually referred to as the background attribute. The VRML 2.0 display construct and background display attribute of one of the grassy knolls shown in Figures 3 and 4 is listed in Figure 5. The background attribute is set as the ambient intensity, diffuse color, and shininess values in the material node of the construct, and corresponds to values set for a DeltaScan Pro GL GenLock system for image combination. Other image combination systems may use different attributes, such as a specific blue background color.

The use of any particular rendering system, such as the Virtual Reality Modeling Language, is not claimed as part of the invention, and the method described will work with any rendering system capable of rendering three-dimensional objects, or two-dimensional projections of three-dimensional objects. VRML is used for virtual object definition in the reference apparatus, rendered through VRML browser software.

The effect of implementing the virtual masking object described in the VRML display construct of Figure 5 is shown in Figure 6. The

grassy knoll of Fig. 5 is approximately 45 meters from the origin of the coordinate system located at the observation point of the user of the invention. By placing the virtual image of a cube beyond the grassy knoll, at 49.5 meters from the origin, and the virtual image of a sphere in front of the grassy knoll, at 42.5 meters from the origin, the masking effect of Figure 6 is obtained, assuming a white background attribute. The actual virtual objects present in the scene, including the grassy knoll's virtual masking object, are shown in Figure 7.

Time Processing Algorithms. Time processing algorithms are used for dynamic scenes in the virtual mask generation step in cases when the range frame rate achieved by the range scanning is not sufficient to provide the illusion of continuous motion (25-30 Hz range frame rate) of the virtual masking objects. With single-beam sequential scanning methods, which are normally used for long ranges of several hundred meters to multiple kilometers to establish a range map, the achievable range frame rate is limited by the speed of light.

As an example, in the laser range map of Figure 3, there are 26068 ranging sectors. A sequential ranging system, such as the laser range scanner used in generating Figure 3, limited at 1500 meters range, has an out-and return time frame of 10 microseconds for the range scanning laser pulse for each ranging sector. In this case this results in a minimum possible frame update time of 0.26 seconds, or a maximum possible range frame rate of 3.83 Hz. Practical considerations of actual ranger electronics implementations make the achievable range frame rate lower than this theoretical limit. The time processing algorithms linearly extrapolate the advance of the vertices of existing virtual masking objects to obtain the intermediate geometry

of these objects between range frame updates obtained by range scanning. An example of the equations implementing time extrapolation for the azimuth A_n, the elevation E_n, and range r_n of a virtual masking object with N vertices is shown in Figure 8. In these equations t is the time for which the extrapolation is performed, T1 is the time of acquisition of the latest range frame, and T0 is the time of acquisition of the previous range frame.

The time processing extrapolation algorithms will be executed normally at a lower frequency than the maximum update rate of 25 to 30 Hz required for the illusion of continuous motion. The extrapolation algorithms can be invoked by control algorithms on an as-needed basis. The logical conditions for invoking and controlling time processing are listed in Table 2. A set of three dimensional "envelope points" (e.g., the ranging sectors of maximum and minimum azimuth and elevation) are selected at the time of acquisition of each virtual masking object for implementing the conditions of Table 2.

Figure 9 illustrates the logic conditions of Table 2. It shows several virtual masking objects, including the apparent terrain observed by range scanning, and other virtual masking objects 131, which include Object A located near to the observation point, and Object B located at the extreme range of range scanning. The envelope points for each virtual masking object are also indicated as black dots (see legend, 132).

The angular position of the envelope points of Object A is assumed to have changed between the last two frame updates as shown by the arrows. If the angular movement of the left-side envelope points of Object A exceeds two ranging sectors (or a minimum number of ranging sectors as specified for a particular

implementation) the extrapolation code will be invoked at least for some of the vertices of Object A by the invocation logic.

The angular position of all of the envelope points of Object B in Figure 9 is assumed to have been less than two scan sectors (or the minimum value specified for the implementation), therefore no extrapolation processing takes place for Object B.

Time processing algorithms do not need to be implemented at all in those applications of the invention in which the observation point of the user of the invention is static, and the real-world field of regard of interest does not contain moving real-world entities.

Although the method of virtual masking object generation is an element of the overall method of the invention, the specific form in which the virtual masking objects are implemented depends on what existing method of virtual object generation and rendering is used in a particular implementation of the present system. The example of Figure 5 uses the Virtual Reality Modeling Language (VRML). Alternatively, a Distributed Interactive Simulation Standard (IEEE 1278-1995) implementation may be utilized. When VRML is used, virtual masking objects are updated through nodes injected by Java code, shown in the computer code listings of Appendix A. When the Distributed Interactive Simulation Standard is used, the virtual masking objects may be generated as Entity Appearance Protocol Data Units, or Dynamic Terrain updates. The method of virtual masking object generation is compatible with any virtual entity generation and rendering method.

Virtual Entity Simulation 107 in Figure 2 is a step of the method performed through existing technologies for computer simulation. Virtual entities may be simulated, by the animation and

control techniques available through the Virtual Reality Modeling Language (VRML). An example of this approach is shown in the listing of Appendix B, in which three battle tanks are animated. Alternatively, the virtual simulation techniques defined in the Distributed Interactive Simulation Standard (IEEE 1278-1995) may be utilized. Virtual entity simulation generates continuously virtual entity appearance data (108 in Figure 2). With VRML, the virtual entity appearance data are updated through animation code implemented as ROUTE constructs, Script nodes, or nodes generated through Java code. When the Distributed Interactive Simulation Standard is used, the virtual entity appearance data are generated as Entity Appearance Protocol Data Units. The invention is compatible with any virtual entity simulation. It is also compatible with a combination of virtual and real entities interacting with the observer, in which case the entity appearance data of real-world entities are transmitted to the apparatus implementing the method by radio or other remote means. In this case, the virtual simulation generates the avatars of the real-world entities, and resolves interactions with the avatars in the same way interactions with virtual entities are resolved.

Virtual Image Generation 109 in Figure 2 renders the virtual entity appearance data generated by the virtual entity simulation and the virtual masking objects. The rendering process itself depends on the particular implementation of the present system. It can be implemented using existing technologies, such as VRML, or the Distributed Interactive Simulation Standard (IEEE 1278-1995), or other, commonly available virtual entity rendering methods. The virtual entity appearance data result in virtual images rendered in their normal appearance attributes, that is in their color, texture, intensity

and other attributes as defined in the virtual entity simulation. The virtual masking objects, and all background of the virtual scene, are rendered in the background attributes compatible with the image combination techniques used in the image combination step. As a result, the virtual image generation step 109 renders masked virtual images, in which those elements that would be masked by natural objects are missing from the images of the virtual entities rendered from the virtual entity appearance data, as shown in Figures 1c and 6.

Image Combination. In this step 110 of Figure 2, the virtual images constructed in the virtual image generation step 109 are combined with the real-world image 114 originating from the video camera 102. The image combination step utilizes commercially available hardware, together with commercial or specially-built software for combining the virtual images with real-world image. The video image will fill out all areas of the display 111 rendered in the background attribute. In the case when the real-world image is provided as an analog video signal, a feasible technique to implement this step is to use a video "GenLock" board, such as a DeltaScan Pro GL used in the reference apparatus, setting the background attribute to 0 intensity, and black color, as shown in the code example of Figure 5. When the real-world image is supplied as digital video, the pseudo-code shown in Figure 10 will implement the appropriate image combination.

Image injection is an optional method of image combination, to be used when the observer 113 would normally observe the real-world scene through an instrument, such as the gunsight of a tank. Two alternatives of image injection are virtual image injection (Figure 11), and combined image injection (Figure 12). In Figure 11, the virtual

image output 140 of virtual image generation is displayed through a color video projector 141. It is then focused at infinity through a collimator 143 and prism 144 and injected into the collimated field of an optical device (such as the gun sight of a battle tank) normally showing only the real-world image 145. The observer 146 using the optical device will see the virtual image over the whole field of view of the real-world image. In Figure 12, the combined image 112 is generated in a color video display projector, which provides the input into an optical device, such as an advanced gun sight that uses video input as its image source.

Display. The output of the image combination step is shown in the display 111 of the combined image 112 in Figure 12, presented to the observer 113. An example of a combined image display, using a VGA monitor in a 16-bit color depth, 640 x 480 pixel display mode is shown in Figure 13. The virtual image of Figure 6, with the grassy knoll's masking object, was combined with a live video scene obtained as an S-video signal from a color video camera. The live scene includes the live original of the grassy knoll, which appears to be masking out a portion of the virtual cube, while the virtual sphere is rendered with no masking.

Alternative display implementations are compatible with the method. These may include flat display panels, image projectors, or virtual reality goggles or visors worn by the observer. In cases when the observer would normally observe the external world through an optical or other instrument, the virtual images can be injected into this instrument, as described under image injection.

The generation of combined images, as described in the method, does not have to reflect what can be seen in the visible

spectrum. Figure 14 shows a combined view in the infrared, as it may show up in the thermal viewer of a modern battle tank or infantry fighting vehicle. The virtual entities generated by a virtual entity simulation may not be those of a real-world object, they may represent fictional characters or entities, or views of conceptual objects. An example of conceptual virtual entities is shown in Figure 15.

In Figure 15, the real-world objects 170 include the terrain, a tree, and a battle tank, in which the observer or observers 171 of a combined reality view presented to them are located. The conceptual virtual entities in the figure are conceptual objects: a path marker 172, a unit boundary 173, a navigational marker 174, and a phase line 175. The present system is required to position the conceptual virtual entities in the display in such a manner that they appear to be present in the landscape visible to the observer(s).

Observer Actions. The observer 113 of the display is assumed to be the user of the invention for various purposes, such as military training, or entertainment. The observer may interact with the apparatus implementing the invention through observer actions 114 in Fig. 2. Observer actions include the observer's interactions with virtual elements in the combined reality scene. An example of such direct interaction is the engagement of a virtual opponent from a combat vehicle, such as a battle tank. In this case, the observer (the battle tank's gunner or commander) would observe and track the virtual opponent embedded in the real-world scene through the tank's actual gun sight, and engage it by firing the tank's main gun. This virtual-live interaction will take place by the observer's actions being captured through the observer's sensors and controls 115 in Figure 2. Observer actions also include interactions of the observer with the real

world, such as head movement of the observer, the observer's use of the battle tank's fire control system controls, or the actual firing of the gun through a trigger. The observer's interactions with the real world also include the movement of the observation point (in the case of a battle tank, the movement of the vehicle and rotation of the turret), and the resulting changes of three-dimensional position and attitude. Observer actions that are relevant to virtual-live interactions may also include the steering of a vehicle housing the observer, or manipulation of control devices such as joysticks or pushbuttons applicable to a specific implementation of the invention.

Observer actions in the virtual-live interoperations include magnification (zoom) control of a simulated battle tank gun sight, the tracking of a specific virtual tank, and engagement of a virtual tank being tracked through the simulated gun sight. They also include scenario selection, scenario start, stop, and re-start. All observer actions are sensed by the various components classified as observer sensors and controls 115.

Observer Sensors and Controls. Observer sensors and controls 115 (in Figure 2) are devices using existing technology, and no specific observer sensor or control is claimed as an element of the invention. The use of observer sensors and controls, however, is an element of the method of the invention. The position and orientation of the observer (including the movement and attitude of the vehicle, if any, housing the observer) is sensed by observer sensors and controls 115 such as a magnetic or infrared head tracker system used in conjunction with virtual reality goggles. Observer controls also sense any signals received directly from the observer, such as changing the optical magnification of a gun sight, or manipulating fire

control system controls. These signals in a combat vehicle may be accessed from an electronic signal distribution system, such as a MIL-STD-1553A data bus, which is present independently of the invention. For moving observers, who may be individuals moving in the real world, or housed in a vehicle, the observer sensors include a position determination system, such as a differential Global Positioning Satellite System (GPS). If the platform housing the observer changes its spatial direction, an attitude sensing system, such as a three-antenna GPS system measuring angular differences by signal phase shift measurements must be included in the suite of observer sensors. Observer's sensor and controls normally include displays or signals resulting from feedback from the virtual simulation, such as virtual battle damage.

In the reference apparatus, GPS survey and optical line-up with surveyed landmarks are used to determine the observer's position and the direction of the scanner and camera optical axes. Observer controls include software-implemented pushbuttons, option selection buttons, and selections from lists of multiple choices, displayed on a color VGA screen, and activated by a mouse control device.

Observer State and Action Processing. The signals generated by the observer sensors and controls are stored and processed by the Observer State and Action Processing step 116 in Figure 2. State vectors (three-dimensional position, angles, velocity, and angular velocities) are generated for the observer, and for the observer's vehicle (if any). Observer actions sensed by the observer's sensor and controls are transformed into data or control signals appropriate for responding to the observer's actions. As an example,

some of the observer actions processed by the present system are listed in Table 3:

Observer action processing may interact with the scanner and camera controls. For example, when the magnification level of the simulated gun sight is changed by the observer, the video camera's 102 zoom level is changed through the observer state and action processing software, by sending a control signal 118 to the camera's electronic control board, which in turn changes the zoom lens position.

Changes in the position and orientation of the observer, of a vehicle housing the observer, or of movable elements of the vehicle (e.g. a tank's turret) as sensed by the observer's sensors and controls are processed by the observer state and action processing step. The virtual entity simulation receives this information for updating the simulation, in which the avatar of the observer, or the avatar of the observer's vehicle is represented. Both the virtual entity simulation and the virtual masking object generation function will respond to the position and attitude changes of the observer, or the observer's vehicle. Such changes will affect the angular orientation of the virtual entities and virtual masking objects, and the apparent distance from observer to virtual entities and masking objects. Changes in the head position of the observer in some implementations of the invention will control the optical axes of the video camera and range scanner, to ensure that scanning and real-world scene acquisition coincides with the observer's current field of view.

When the observer's sensors and controls include signals acquired from a vehicle, such as the fire control system information acquired through a data bus in a battle tank housing the observer, this information is also processed in this step. For example, in computing whether a virtual opponent was actually hit by fire from the host

vehicle's main gun, the fire control system data available from the vehicle bus are utilized. The result of such calculations is then fed to the virtual entity simulation to provide damage information.

Information fed back from the virtual simulation is also processed in the observer's state and action processing step. For example, if one of the virtual tanks represented in this simulation engage the avatar of the observer's tank, the resulting damage information will activate damage displays, or a "killed" indicator in the observer's sensors and controls suite.

Reference Apparatus

Figure 16 is a simplified block diagram of the reference apparatus showing one possible implementation of the present invention. The real-world scene 216 is observed through a video camera 200, and it is range scanned with the laser scanner. The video camera is a commercially available remote-control camera. The video camera can be moved in pitch and yaw (pan motion), and also zoomed through a nominal magnification range of 1.55x through 10x magnification. These functions are activated by the digital control signals sent to the camera's internal control board through an RS-232 camera control cable 202. The internal control board, in response to these signals, activates the camera's zoom, pitch, and pan motors. Through built-in sensors, the camera also senses the current zoom, pan, and pitch values, which are used by the internal control board as feedback signals, and are available through the camera control cable 202 for external processing. The video signal, in the S-video analog format, is acquired from the camera through the video cable 201, which connects to the video mixer through its S-video input port.

The laser scanner 203 is a modification of a commercially available solid-state infrared surveyor's ranging laser, operating at 905 nanometer wavelength, sending out 40 nanosecond repeated laser pulses at 47 watts peak power. Range is determined by measuring the time differential between start of a pulse, and the start of the reflected return signal, separated from ambient light by a narrow band filter. Modifications made to the laser unit's electronic control board made by the manufacturer have set the pulse rate at 266Hz for range scanning use. The laser, in normal surveying operations, without the use of retro-reflectors, has a maximum range of approximately 1200 m. This range is reached by processing multiple returns from the same target, and utilizing a set of logic functions which disregard uncertain returns. The laser has separate sending and receiving apertures. The scanner elements are built from commercially available components, as shown in the schematics of Figures 17 and 18.

The transmitter section of the laser scanner, shown in Figure 17, reflects the laser beam 228 originating from the laser diode 220 and focused through the lens system 221 of the commercial laser unit through a horizontal scan mirror 222 and a vertical scan mirror 225. The horizontal scan mirror is mounted on a shaft, which rotates around the horizontal scan axis 224, driven by the horizontal scan motor 223 in steps equal to 1/2 of the horizontal angular width of the ranging sector (approximately 0.49 milliradians). The horizontal scan mirror doubles the relative incidence angle of the laser beam, resulting in horizontal scan steps incremented by horizontal angular width of the ranging sector. The direct current stepping motor is controlled through the electronics unit 205 in Figure 16, which steps the motor 266 times per second, coordinated with the laser pulses fired by the ranging laser. Time marks corresponding to the laser firing signals are

transmitted through the RS-232 scan control cable 204 from the laser's built-in electronic control board to the electronics unit of the apparatus 205. The electronics unit, in turn, commands the steps of the horizontal scan motor. The timing of the electronics unit commands to the scan motor is generated so that it precedes the next laser pulse firing by approximately 15% of the cycle time between laser pulses, providing adequate settle time for the horizontal scan mirror. The horizontal scan mirror is a 15 mm diameter circular mirror, controlling a laser beam of approximately 8 mm in diameter, and expanding at an angle of 0.98 milliradians. The horizontal scan mirror performs a linear nodding motion about the horizontal scan axis. After completing a horizontal scan line of 266 steps, the next scan line proceeds in the opposite direction after the laser beam's elevation is changed by the vertical scan mirror. The resulting scanning pattern is shown in Figure 19.

The vertical scan mirror 225 is stepped by half of the vertical angular height of the ranging sectors (approximately 0.49 milliradians) upon the completion of each horizontal scan line. The vertical scan mirror, driven by the vertical scan motor 226 is mounted on a shaft that rotates about the vertical scan axis 227, which is perpendicular to the horizontal scan axis. The steps of the vertical scan motor are controlled by the electronics unit 205 through the scan control cable 204. Each vertical step is activated in response to an opto-mechanical interrupter connected to the horizontal scan mirrors. At the end of the scan line, the mechanical element of the interrupter interrupts a light beam incident on a commercially available photo-electronic sensor, which in turn interrupts an electronic signal sent to the electronics unit through the RS-232 control cable. The elevation of the outgoing laser beam 228 is raised by an angle corresponding to

twice the angular step of the vertical scan mirror, resulting in the scanning pattern shown in Figure 19. The control logic of scanning is summarized in Table 4.

The receiver section of the laser apparatus, shown in Figure 18, is closely integrated with the transmitter section. The incoming laser reflection 233 is reflected through the vertical scan mirror 232, and the horizontal scan mirror 231, into the sensor cell 230 of the laser electronics unit. The vertical scan mirror and the horizontal scan mirror is mounted on a shaft rotating about the vertical scan axis 227. This is the same shaft, rotating about the same axis as the one mounting the vertical scan mirror of the transmitter section. The angles of vertical scan mirrors of the transmitting and receiving sections are therefore always at the same angle, except for dynamic transients of shaft rotation at their respective mounting points, resulting from the shaft's response to the motor steps. The horizontal scan mirror of the receiver section is also on a common shaft with the horizontal scan mirror of the transmitter section, rotating about a common axis 224, driven by the same horizontal scan motor 223. The lens system 234 and the sensor cell of the receiver section are standard elements of the commercial laser range finder used in the laser range scanner. The distance of the object causing the laser reflection is measured by the electronics section from the time difference between firing the outgoing laser beam 228 (in Figure 17), and the laser reflection.

The laser scan pattern of the reference apparatus is shown in Figure 19. Six horizontal scan lines are shown. Each scan line is composed of 266 scan steps. The arrows on the scan lines show the direction of advance of scanning for each scan line. The scan frame for this implementation is defined as the 98 successive scan lines

implemented by the apparatus. Successive scan lines proceed from the highest azimuth at the top of the scan frame to the lowest azimuth at the bottom of the scan frame. Upon completion of the scan frame the scan starts with a new scan frame, at the top of the scan frame.

The laser range data signals, resulting from each laser pulse-reflection sequence, are delivered through the standard RS-232 laser signal cable 206 to the laser scanner control computer 208. These signals may be, at the option of the operator, displayed by the laser scanner control computer in Display A 209 in Figure 16, (e.g., a VGA display) as range maps, such as shown in Figure 3. The laser range data signals received by the laser scanner control computer are re-formatted to the laser data format of the reference apparatus for transmission to the virtual simulation computer 210 through the RS-232 laser data cable 217. Use of the laser data format ensures that with change of brands in the laser range scanner's laser unit, there will be no change in the format of laser data received by the virtual simulation computer. The laser data format is defined in Table 5.

Start and stop commands for starting up the laser are transmitted to the laser range scanner directly from the laser scanner control computer through the RS-232 laser signal cable 206. Scanner motor start and stop commands are transmitted from the laser scanner control computer through the RS-232 interface control cable 207 to the electronics unit 205, which executes the stepper motor control programs. These programs, in turn, generate the detailed stepper motor control signals sent to the laser range scanner through the scan control cable 204.

Virtual masking object generation is presently implemented by virtual masking object (VMO) generation software running in a virtual simulation computer 210 (e.g., a personal workstation running two 266

Mhz Pentium II processors). The processing flow of the VMO generation software is indicated in Figure 20. Appendix A contains the listings of the key software routines implementing this processing flow for the reference apparatus.

The laser ranging data, formatted in the laser range data format (Table 5), are received by the virtual simulation computer 210 through a serial port, and are processed by a standard communications processing module 241. This module, and other standard modules normally available through various vendors, requiring only minimal customization within the capability of any programmer familiar with the C and C++ languages and standard communications data processing, are outlined in dashed lines in Figure 20 and are not listed in Appendix A.

The communications processing module 241 places the laser ranging data into a global memory buffer module 242. A global memory interface module 243 fetches the laser ranging data for each consecutive ranging sector for processing by the virtual blade object generation module 244. This module agglomerates adjacent ranging sectors on the same scan line into virtual blade objects, by implementing the rules for virtual blade object generation shown in Table 1. The processing generates an array of virtual blade objects, which are then processed by the virtual masking object (VMO) generation module 245, which implements the VMO generation rules of Table 1. The data arrays defining the border points of the VMOs are then read by the VMO VRML construct generation module 246, which assembles data strings describing the individual VMO. An example of VMO strings generated by this module is shown in Figure 5. These VMO strings are passed by the VMO VRML construct generation module into the current virtual scene for VMO rendering 247. The

current virtual scene includes the virtual entities generated by virtual entity simulation (Figure 22).

The image of the current virtual scene, together with the image of the surrounding control frame (see Figure 21), jointly form the computer-generated image 248, that may be optionally displayed in Display B 211 in Figure 16. The control frame is implemented in HTML. The computer-generated image is transmitted to the video mixer 213 through the virtual display cable 212 (e.g., a VGA/SVGA computer display cable).

The control frame shown in Figure 21 is shown above the virtual scene in the computer-generated image. It is implemented by the Java and VRML-capable browser running the Java applet code for VMO VRML construct generation, and rendering the virtual scene, as indicated in the code listings of Appendix B.

Virtual entity simulation is implemented in the virtual simulation computer through the virtual simulation and control software. The processing flow of the virtual simulation and control software is shown in Figure 22. Appendix A contains the listing of the key modules of the virtual simulation and control software.

The observer's command interface is implemented in the HTML control frame 251 (shown in detail in Figure 22) through the various selection lists and control buttons. The observer's control actions result in the execution of Javascript command messages 252. The Help button in the control frame invokes a stand-alone, floating HTML help page, and the Tech Panel button invokes a separate control frame for adjusting Display B. These two controls are not relevant to the method disclosed, and their functions are not discussed further, except to remark that the Tech Panel controls implement boresighting of the virtual, video, and laser scenes. The Select Scenario list, and

Start and Reset buttons in the control frame execute control function related to the virtual simulation only. The code sequences resulting from activating these controls are listed in Appendix B. Select Scenario selects one of several available scenarios for the behavior of the virtual entities being simulated (in this case T-80 main battle tanks). Other scenarios not related to tanks, or military training, can be selected by a user of the reference apparatus after minor modification of the VRML data, and the labels of the scenario control list. The Start and Reset buttons start and reset a selected scenario.

The Zoom Control list box, the AutoTrack check box, and the Fire button in the control frame emulate some of the functions of the controls available to the gunner of a main battle tank, or to the gunner in a tank simulator, for engaging the virtual T-80s operated by the virtual simulation.

Activating any of the controls available in the control frame, a Javascript command message 252 is executed, which in turn, invokes the command transfer 253 functions in a Java applet. In the case of the Zoom Control, the command transfer function executes the code in a dynamic link library (DLL) for camera commands 255, and also a VRML script function for VRML command execution 254. The camera command functions ensure that the scene displayed by the video camera and the VRML-generated virtual scene are displayed at the same magnification. All other control objects in the control frame interact only with VRML command execution. The VRML command execution script functions, in turn, activate changes in the Virtual Animation and Rendering 256 elements implemented in VRML. The virtual animation and rendering function, in turn, renders and displays the virtual elements of the computer-generated image 248, which also contains the virtual masking objects.

The computer-generated image 248 may be optionally displayed in Display B 211 in Figure 16. The computer-generated image is transmitted to the video mixer 213 through the virtual display cable 212 (e.g., a standard VGA/SVGA computer display cable). The virtual entities implemented by the virtual simulation and control software are capable of engaging the avatar of the observer's vehicle, depending on the scenario loaded in. In the reference apparatus, in which the observer's position is assumed to be static, the avatar of the observer's vehicle is implicitly located at the observer's VRML Viewpoint node coordinates. The VRML route nodes describing the motion of the virtual entities are generated automatically by a route generation 257 module (written in Visual Basic and Java). The route generation module is regarded as a support module of the visual simulation, generally implementing existing technology, and will be described separately.

Image combination is implemented through the video mixer 213 (in Figure 16), a commercially available DeltaScan Pro GL GenLock unit. The S-video format analog video image generated by the video camera is delivered to the video mixer by means of the video cable 201, a standard and commonly available S-video cable. The computer-generated image resulting from the virtual masking object generation and virtual simulation and control functions is a standard VGA or SVGA color image generated with 16-bit or 24-bit color pixels (depending on settings in the virtual simulation computer). This image is delivered to the video mixer through a standard, commonly available VGA/SVGA cable. The background color used in the reference apparatus is black (0,0,0 RGB values), with 0 intensity. The video image shows up as the background image, that is, in all areas of the computer-generated image rendered with these attributes. This

results in the combined reality scene showing virtual entities in the real world, the objective of the invention.

The output of the video mixer is an S-video signal, available through the S-video feed cable 214 (in Figure 16), a standard S-video cable. The available settings of the video camera and the video mixer also permit the use of a standard NTSC video signal, in which case the S-video compatible cables are replaced by a standard NTSC video cable. The display function of the combined reality scene in the reference apparatus is implemented through an S-video display.

The observer of the reference apparatus is at a static position, observing the combined reality scene through a simulated gunsight of a main battle tank. In the reference apparatus observer controls are those on the control panel (Figure 21). A Global Positioning Satellite (GPS) receiver, with optional differential correction, and an aircraft altimeter determine the three-dimensional position of the observer. Alignment of the apparatus with the vertical in is assured through two bubble levels. The azimuth of the boresight axis is determined with the help of landmarks, a telescope attached to the laser scanner housing, and a calibrated compass azimuth ring. Observer action processing takes place through the software responding to control panel actions. Generation of geographical information and virtual entity movement and alignment information takes place through the use of a route generation module, operated in advance of using the apparatus at a given route generation module.

The route generation module 257 is an inter-operating collection of software written in Visual Basic and Java. It supports the virtual simulation and control, and observer sate and action processing functions. It is a supporting module of the reference apparatus, which can be implemented without operating this module.

Only the summary of the operation of the route generation module is described, on the basis of which an equivalent software module can be developed by a programmer familiar with its implementation languages, VRML, and the US Geological Survey Digital Elevation Model (USGS DEM). The VRML route nodes describing the motion of the virtual entities are generated automatically by the module. The terrain preparation component of the module reads in USGS DEM digital terrain files 258 that include the observer's location, and generates a VRML elevation grid from the USGS DEM data. The route builder component of the module automatically prepares three-dimensional VRML position and orientation interpolator nodes for the scenario from the VRML elevation grid and routes overlaid on a map display by the observer. These nodes are read at simulation start time by a Java command transfer function, which injects them into the VRML content of the virtual simulation (implemented through the VRML command execution, and virtual animation and rendering functions).

The VRML elevation grid derived from the USGS DEM data usually does not correspond exactly to the terrain visible through the video camera 200 (Figure 16), or sensed by the laser scanner 203. To correct the VRML elevation grids on the basis of real-world data, the Virtual Masking Objects (VMOs) built by the VMO VRML construct generation module 246 (in Figure 20), and digital scene files 259 can be read by the correction component of the route generation module. The digital scene files are video snapshots received from the video camera and saved as standard JPEG or BMP files. Alternatively, during scenario preparation, before a scenario is executed by the virtual simulation and control software, the direct video feed from the video camera can be used to edit VRML terrain grid. In this editing

mode, the virtual image of the terrain grid, displayed by the Java component of the route generation module, is mixed, through the video mixer 213 with the video image of the corresponding terrain in the S-Video display 215. The controls of the route generation software in this case permit the direct manipulation of each point on the terrain grid.

The range of the ranging sectors within the field of regard of a range scanning device may be determined by other means besides laser scanner devices as described for the reference apparatus. As a design guideline for optical or electromagnetic ranging, such as microwave ranging, the angular extent of the ranging sectors in radians is limited by the ratio of wavelength to transmitter aperture (reflector, antenna, or lens). This limitation also applies to acoustic ranging.

Embodiments of the invention using step-by-step range scanning methods, advancing from ranging sector to ranging sector, such as shown in the diagram of Figure 19, require a minimum frame scan time of $t = 2nr/c$, where n is the number of ranging sectors in a scan frame, c is the speed of light for electromagnetic scanning and the speed of sound for sonic scanning, and r is the maximum range of the scanning instrument. When t is greater than approximately 0.03 seconds, the virtual masking object time processing extrapolation methods, such as shown in Fig. 8, must be invoked for dynamic real-world scenes. The relationship $r = 0.015c/n$ will determine the maximum range for a given scanning frame size without applying virtual masking object extrapolation. For ranging sectors corresponding to the pixels of a VGA display, $n = 288{,}000$, limiting the value of r to 15.6 meters with electromagnetic scanning. These relationships also mean that sonic scanning of dynamic scenes, in

essence, will always require the use of extrapolation algorithms. The limiting factor in the case of sonic scanning in dynamic scenes is the scan frame repetition rate. Estimating the minimum acceptable ranging sector size as approximately 1 degree in azimuth and elevation, the scan frame repetition rate is approximately 0.5 Hz at a 4-meter range. Scan frame repetition rates below this value are unlikely to provide adequate information for realistic extrapolation.

Step-by-step ranging using electromagnetic scanning can provide high-resolution virtual masking for longer ranges without extrapolation when the optical axis of the range scanner is coupled to a sensor tracking the foveal motion of the observer. Such sensors are available in experimental and military systems. The foveal zone is less than 3 degrees in angular extent. It is the only zone within which fine details are observed by the human eye. Considering a one milliradian ranging sector, foveal motion-coupled optical or electromagnetic range scanning can be performed without the need for extrapolation up to a range of approximately 1700 meters. A second scanner, operating at a different wavelength, may then cover the whole visual field at the same range without the need for extrapolation with ranging sectors of approximately 1 degree of angular extent.

High frame rate sequential scanning requires high-frequency beam steering methods. Using a two-mirror arrangement similar to that of Figure 17, and with the ranging sector and field of regard of the reference apparatus, the required horizontal mirror rotation frequency is approximately 100 Hz. This can be achieved by mechanical means. Considerably higher frequencies can be achieved by laser-transparent crystals excited by alternating electrical fields (Figure 23). Beam deflection amplitude by this method is limited approximately to ± 1.5 degrees. The limited scan angles of the various high-frequency beam

steering methods can be multiplied by reflecting mirrors, each mirror multiplying the angle by a factor of two.

Active flash ranging methods are based on illuminating a scene simultaneously with ranging pulses covering the whole field of view of interest, and receiving the return reflection through a lens system directing the reflection to a CCD sensor matrix device. The practical embodiment of laser ranging devices built using this method used additional range discrimination besides return time discrimination, such as multiple amplitude-modulated ranging flash pulses, or frequency-modulate ranging flash pulses. Flash ranging permits the generation of virtual masking objects without extrapolation. The limitation of this method is in its large power requirement for large fields of regard, which, in turn, limits its practical range to around 100 meters.

Passive ranging methods are based on parallax measurements of ranging sectors (pixels) in coaxial CCD matrix cameras with parallel optical axes separated by a known baseline (Figure 24). The identification of ranging sectors may take place by comparison of individual pixel color and intensity values, or by identifying each pixel through its return reflection of a laser scanning beam through a hybrid passive-active ranging approach. In terms of the notation of Figure 24, the range d from the passive range measurement device is $d = a/(\tan X1 - \tan X2)$, or for $d >> a$ $d = a/(X1-X2)$ where X1 and X2 are the apparent angular increments of the pixel of interest from the optical axes of Camera 1, and Camera 2, respectively. The method is compatible with high-speed, parallel hardware level processing.

Most of the examples discussed in the disclosure are in the area of military training, enabling individuals or combat vehicle crews to engage simulated opponents in a real-world environment. The

invention is not limited to such applications; it can be applied with equal efficacy to inserting virtual elements into films in an automated manner, displaying the appearance of potential construction in its planned exterior or interior site, or as an interactive, recreational device. It can be used as an interior decorating or arrangement planning and demonstration aid, or as an aid showing the potential effects of reconstructive surgery or dental work. The method disclosed is capable of supporting such applications, or other implementations in which virtual, computer-generated scenes must appear to fit seamlessly into a real-world environment.

SYSTEM FOR COMBINING VIRTUAL IMAGES WITH REAL-WORLD SCENES

ABSTRACT

A system for combining virtual images with images of the real world. A range scanner determines the shape and distance of real-world objects within a field of regard of interest to the observer using the apparatus. Virtual masking objects, which are simplified computer models of real-world objects, are derived from the range data. Virtual entities that are not present in the real world are simulated in an electronic computer. Images of the virtual entities and the virtual masking objects are combined into masked virtual images. The masked virtual images show the portions of virtual entities that would be visible if these entities actually existed in the real world. The masked virtual images and images of the real world scene are combined in such a manner that the virtual images appear to be obscured, when appropriate for their simulated, virtual location by real-world objects. The resulting combined image is presented in an electronic or optical display.

Glossary of Terms

avatar	A virtual image and supporting virtual object data and programs representing a real-world entity in virtual reality
combined reality, combined reality scene	Apparent reality, with virtual elements projected into the image of the real world. Virtual and live elements may interact through simulation.
live original	An entity in the real world, represented by an avatar.
masked virtual image	That portion of a virtual image which would be visible from a given observation point if the virtual image represented a real-world entity.
range frame rate	The update rate of a range map
range map	Array of the ranging sector ranges measured in a field of regard
ranging sector	An angular space within which range to an object is measured
real-time virtual image obscuration	Virtual image obscuration taking place at a fast enough rate (usually 25 to 30 display frames per second) to present apparent continuous motion and obscuration of virtual entities by real-world objects.
real-world entity	An actual object, living being, or phenomenon
real-world coordinates	A coordinate system for referencing real-world entities
real-world scene	The set of real-world entities which may be observed by, interacting with, or used by the user of the invention
state vector	The three-dimensional position, orientation, velocity, and other descriptors of the current state of a virtual or real-world entity
virtual coordinates	A coordinate system in which a simulation represents virtual entities. In combined reality is coincident with the real-world coordinates.
virtual entity	An object, living being, or phenomenon represented by simulation
virtual image	The image of a virtual entity
virtual-live interaction	The interaction of virtual entities with real-world entities

virtual masking object	A virtual object implementing the obscuration of objects caused by a real-world entity, or by a part of a real-world entity, as seen from the current point of view of an observer.
virtual position	Location in the real world where a virtual entity is represented
virtual reality	An artificial world, created in a digital computer, in which various virtual entities seem to exist, operate, and interact with each other.

Function	Condition
Define zero, one, or more coherent virtual objects within each scan line, having an angular height of one ranging sector (objects referred to as virtual blade objects).	Add new ranging sectors to the current virtual blade object as long as the individual ranging sectors fall on a straight line within tolerance "d0" (for example, 1% of the range, and no less than the scanner's ranging error) as computed by a least-squared error algorithm.
Identify the virtual blade objects that belong to a common virtual masking object .	Add new virtual blade objects to a virtual masking object as long as the end points fall into a common plane within tolerance "d1" as computed by a least-squared error algorithm, and the virtual blade objects added are adjacent virtual blade objects already within the virtual masking object.
Filter the edges of virtual masking objects to remove unnecessary vertices.	Build straight line edges for the virtual masking objects, remove intermediate vertices which fall on a straight line edge within tolerance "d2" as computed by a least-squared error algorithm
Limit virtual masking object size to that appropriate for fast rendering.	Limit virtual masking objects to "n" (for example, 50 or 100) vertices.
Remove holes from virtual masking objects.	Fill in spaces of one ranging sector between adjacent virtual blade objects if the virtual blade objects otherwise fall into a common plane.

Table 1: Example of rules to generate virtual masking objects.

Function or Action	Condition
Do not activate time processing	Observation point is static and there are no moving real-world entities in the scene
Time processing frequency for any vertex of any virtual object. Assure that vertex moves if movement is at least one, or a few, ranging sectors.	Number of ranging sectors swept by movement of nearest envelope point in each frame update cycle, divided by a heuristic factor (e.g. 2), limited by maximum update rate (e.g. 30Hz).
Decision to re-render a virtual masking object	Only if one or more vertices are re-computed by time processing.

Table 2: Time processing algorithm invocation logic.

Observer Actions	Effect of Observer Action	Implementation and Reference
Tracking of Virtual Targets by selecting Autotrack button	Simulated gunsight reticle placed on target, reticle automatically follows target.	Autotrack software sequence in Appendix B.
Fire gun at virtual target	Engage target being auto-tracked	Engage software sequence in Appendix B.
Change gunsight magnification	Change simulated gunsight magnification	Zoom change software sequence in Appendix B.
Select, start, pause, stop scenario	Selects and controls a scenario in virtual simulation	Scenario control software sequence in Appendix B.

Table 3: Examples of observer action processing

Scanner Action	Control Logic
Laser pulse fire	266 Hz repetition rate
Increment horizontal scan angle (azimuth)	Follow laser pulse firing signal with sufficient time, as a minimum, for laser out-and return at maximum range. Precede laser signal with sufficient settle time for step motor-mirror subsystem
Terminate horizontal scan line	Respond to opto-mechanical interruptor
Increment vertical scan angle (elevation)	Follow termination of horizontal scan line
Initiate new horizontal scan line	Follow elevation increment, reverse horizontal scan direction
Terminate scan frame	After last horizontal scan line complete
Initiate new scan frame	After termination of frame, if new frame start command is received

Table 4: Laser Scanner Control Logic

Data	Implementation
Serial Communication Parameters:	14400 Baud, No parity, 8 bits, 1 stop bit
Start frame, upper left corner of scan frame	"BFU"
Start frame, lower right corner of scan frame	"BFD"
Range value scaling for frame starting now	"SC = x.xxx"
Start next scan line No. ddd	"SLddd"
End of current frame	"EF"
Range data following "SLddd"	Scaled binary range of two bytes for each measurement. Both bytes have a "Byte Indicator Bit" at the high bit: raised (1) for the high byte, low (0) for the low byte.
Positional values of range data bytes, providing a numerical range of 0 - 16383	"Low" byte 64 32 16 8 4 2 1 "High" byte 8192 4096 2048 1024 512 256 128
Application of Range Scaling Factor	Multiplier for the values transmitted by the range data bytes.

Table 5: Laser Data Format

Step or Element of the Method of the Invention	References	Implementation in the Reference Apparatus	References
Range Scanning	101 in Figure 2	Laser Scanner (203), Electronics Unit,	Figs. 16, 17, 18 and 19, Tables 4 and 5
Video Camera	102 in Figure 2	Video Camera (200)	Figure 16
Virtual Masking Object Generation	105 in Figure 2, Figures 3-9, Tables 1-2	Virtual Masking Object Generation Software in Virtual Simulation Computer	Appendix A, 210 in Fig. 16, Fig. 20
Virtual Entity Simulation	107 in Figure 2	Virtual Entity Simulation and Control Software	Figures 21-22
Virtual Image Generation	109 in Figure 2	Virtual Reality Modeling Language, VRMLScript	Figure 22 (254, 256)
Image Combination	110 in Figure 2, Figs. 10-15	DeltaScan Pro	Fig. 16 (213)
Display	111 in Figure 2	S-Video Display	Fig. 16 (215)
Observer's Sensors and Controls	115 in Figure 2	Control Frame, Global Positioning Satellite rcvr.	Figure 21
Observer State and Action Processing	116 in Figure 2, Table 3	Virtual Entity Simulation and Control Software	Appendix B

Table 6: Correlation of the Method of Invention with the Reference Apparatus

Appendix A - Software Code for Virtual Masking Object Generation

The code segments presented in this Appendix, written in the C and Java languages will enable a programmer experienced in the above languages and in the Virtual Reality Modeling Language to reconstruct the Virtual Masking Object Generation elements of the software supporting the reference apparatus, and to build other software to implement virtual masking object generations in applications other than the reference apparatus. Standard communications routines, such as WriteTTYBlock(), have been ommitted for brevity, as these routines are available in various textbooks and code examples supplied by major software builders.The comment "Omitted, standard software" is inserted after the function declarations in such cases.

This Appendix contains the following sections Page No.

1 Virtual Masking Object Generation - C Source Code 2
2 C-Java Interface Code 36
3 Java Code to Generate Virtual Masking Objects 42
4 Java Code to Get Laser Data Stream 45

Meta Exhibit 1011
Meta v. Mullen - Page 0047

1 Virtual Masking Object Generation - C Source Code

```
#define DEBUG TRUE

//
//  Module: vmo_gen.c
//          Copyright Pathfinder Systems 1997
//  Description of functions:
//
//

#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "tty.h"
#include "ttyrange.h"
#include <windows.h>
#include <memory.h>
#include <malloc.h>
#include <windowsx.h>

#pragma warning(disable: 4001)      /* Single-line comment */

#include <tchar.h>
#include <errno.h>

#define ALLOWED_ITERATION_ERROR 10
#define MIN_VMOSIZE 20
#define MAXVMO_DEPTH 20
#define EOF_REACHED  31415//random number
#define PPortBase 0x378
#define PORT_A (PPortBase)
#define PORT_B (PPortBase + 1)
#define PControl (PPortBase + 3)

#define scolor(r,g,b)  (((b&0xff)<<16) | ((g&0xff)<<8) | (r&0xff))

COLORREF PMap[1650];

#define MAX_COLORS (256 * 5)
#define LIGHT_GREY (0x808080)
#define BLACK      (0)
#define WHITE      (0Xffffff)

#define ALLOWED_ITERATION_ERROR 10
#define MAPSIZE 10000000L
#define NUMOBJECTS 30
#define CURRENTLY_WRITING     1
#define READY_TO_READ         2
#define CURRENTLY_READING     3
#define SUCCESSFULLY_READ     4

#define RADS_PER_PIXEL (float)(0.00095)
```


Meta Exhibit 1011
Meta v. Mullen - Page 0048

```
#define EL(i) ((NUM_ROWS/2 - (i+1))*RADS_PER_PIXEL)
#define AZ(j) (((j) - NUM_COLUMNS/2)*RADS_PER_PIXEL)
#define RANGE(i,j) (RangePtr->RngFrame[i*NUM_COLUMNS + j] * RangePtr-
>scale_factor)

#define CAMHEIGHT (1.6)
#define XCOORD(az,el,d) (sin(az) * cos(el) * (d))
/* note: VRML looks in the -Z direction ==> all Z coords are negative
*/
#define ZCOORD(az,el,d) (cos(az) * cos(el) * (d))
#define YCOORD(az,el,d,z) ((sin(el) * cos(az) * (d)) + CAMHEIGHT -
sin(RADS_PER_PIXEL) * (z))
#define HALF_LASER_PIXEL (float)(RADS_PER_PIXEL/2.0)
typedef struct IFS_Point{
            int         x;
            int         y;
            int         z;
};
typedef struct VMO_PolyStruct{
      int         num_indices;
      struct IFS_Point *vertices;
};
typedef struct VBO {
      int         blade_num;
      int         row_index;
      int         start_x;
      double      start_z;
      int         end_x;
      double      end_z;
      double  mean_range;
} Vbo;
typedef struct RangeMatrix {
      float range;
      int status;
};
typedef struct V_Point{
      int         startcolumn;
      int         startrow;
      float startrange;
      int         endcolumn;
      int         endrow;
      float endrange;

};
typedef struct VMO_Struct{
      int num_blades;
      struct V_Point *vertices;
};
struct VMO_PolyStruct *VMO_Polygons;
struct VMO_Struct *VMO_List;
struct RangeMatrix VMO_Matrix[NUM_COLUMNS][NUM_ROWS];
struct RangeMatrix VMO_Matrix_Temp[NUM_ROWS][NUM_COLUMNS];

Vbo     VBO_List[ NUM_ROWS * NUM_COLUMNS / 2];
int num_blades = 0;
int total_VMOnum;
HWND Hwnd_Offset;       /* pointer to Offset Cal Window */
```


Meta Exhibit 1011
Meta v. Mullen - Page 0049

```
#define DEFAULT_OFFSET 1
double range_at_cursor = 0.0;
double initial_range = 0.0;
int is_started = 0;
FILE *fd_COM2, *fd_LPTR1;
int num_cells, num_cells2 =0;
clock_t start, finish;
double  duration;
char buffer[50];
char SharedMapName[50] = "VictorRangeMap";
HANDLE hVictorMap;
void * lpVictorMap;
void * lpVictorMapRead;
int Process_chars(char *abIn, int nLength);
int test_process_chars( void );
int test_process_file( void );
int make_vbo_list( HWND hWnd);
int old_make_vbo_list( void );
Vbo * FindBlade( int ix, int iy );
int make_blades_to_objects( HWND hWnd );
int polygon_iteration_down(int i, int last_set_vertex, int
current_vertex);
int polygon_iteration_up(int i, int last_set_vertex, int
current_vertex);
int polygonize_VMO(HWND hWnd);
int idd_save( void );
int sendno(int no);
float getRange(int x, int y);
int isZmatch(int x1, int y1, int x2, int y2);
int ScanL(int xl, int y, int seedval, int fillval);
int ScanR(int xr, int y, int seedval, int fillval);
int OpenVMO(int VMOnum);
int find_bladeend(int y, int fillval);
int find_bladestart(int y, int fillval);
int lineavailable(int y, int fillval);
int num_bladesinline(int y, int fillval);
int SeperateVMO(int X_seed, int Y_seed, char Direction,int prev_XL, int
prev_XR, int seedval, int fillval);
int FindVMO(int X_seed, int Y_seed, char Direction, int prev_XL, int
prev_XR, int seedval, int fillval);
void printstatusmatrix(void);
int inrange(float rangedata);
int Load_Resource(HWND hWnd);
void Load_SharedMem(void);

HANDLE GetMapHandle(char *MapName, long mapsize);

NPRANGEMAP RangePtr;

#define HUGE_BUFFER_SIZE 65536        // bc debug
BYTE Huge_Buffer[HUGE_BUFFER_SIZE];   // bc debug
BYTE * Huge_Ptr = Huge_Buffer;        // bc debug
int sequence_errors           = 0;
int line_range_errors    = 0;
int col_range_errors     = 0;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0050

```
//-----------------------------------------------------------------------
------
//  int PASCAL WinMain( HANDLE hInstance, HANDLE hPrevInstance,
//                      LPSTR lpszCmdLine, int nCmdShow )
//
//  Description:
//     This is the main window loop!
//
//  Parameters:
//     As documented for all WinMain() functions.
//
//-----------------------------------------------------------------------
------



int PASCAL WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow )
{
      MSG   msg ;
      NPRANGEMAP local_range_ptr;

   /*  load false color map */

   long rr,gg,bb, indx;
   int fInit;

   rr = 0xff;
   indx = gg = bb = 0;

   for (gg = 0; gg < 0x100; gg++){
            PMap[indx++] = scolor(rr,gg,bb);
   }
   gg = 0xff;
   for ( ; rr > 0 ; rr--) {
            PMap[indx++] = scolor(rr,gg,bb);
   }
   for ( ; bb < 0x100; bb++) {
            PMap[indx++] = scolor(rr,gg,bb);
   }
   bb = 0xff;
   for ( ; gg > 0 ; gg--) {
            PMap[indx++] = scolor(rr,gg,bb);
   }
   for ( ; bb > 0 ; bb--) {
            PMap[indx++] = scolor(rr,gg,bb);
   }

#define MAX_COLORS (256 * 5)

   /* Create a named file mapping object */
      // sigrec_t *SIG_REC_P;
      // pointer to the memory mapped signal structures
```


Meta Exhibit 1011
Meta v. Mullen - Page 0051

```
    local_range_ptr = (NPRANGEMAP)CreateFileMapping(
        (HANDLE) 0xFFFFFFFF, /* use paging file    */
        NULL,                /* no security attr.  */
        PAGE_READWRITE,      /* read/write access  */
        0,                   /* size: high 32-bits */
        sizeof( RANGEMAP ),  /* size: low 32-bits  */
        "BruceRangeMap");   /* name of map object */
    if (local_range_ptr == NULL){
        return FALSE;
      }

    /* The first process to attach initializes memory. */

    fInit = (GetLastError() != ERROR_ALREADY_EXISTS);

    /* Get a pointer to the file-mapped shared memory. */
      /* please note that SIG_REC_P will probably be different */
      /* from process to process, but may be same if no other */
      /* resources are allocated elsewhere, but don't count */
      /* on it!! */

    RangePtr = MapViewOfFile(
        local_range_ptr,       /* object to map view of    */
        FILE_MAP_WRITE, /* read/write access        */
        0,              /* high offset:   map from  */
        0,              /* low offset:    beginning */
        0);             /* default: map entire file */
    if (RangePtr == NULL)
        return FALSE;

    /* Initialize memory if this is the first process. */

/*    if (fInit)
        memsetnoblock(local_range_ptr, '\0', sizeof(sigrec_t) *
              MAX_THREADS_FOR_SIGNALS); */

/* set in a temporary picture, until real one loaded */

      RangePtr->rows = NUM_ROWS;
      RangePtr->cols = NUM_COLUMNS;
      RangePtr->scale_factor = 1.0;

      for (rr = 0; rr < NUM_ROWS; rr++) {
            for (gg = 0; gg < NUM_COLUMNS ; gg++ ) {
                  RangePtr->RngFrame[rr*NUM_COLUMNS + gg] = (5*(gg +
rr))%MAX_COLORS;
            }
      }

   if (!hPrevInstance)
      if (!InitApplication( hInstance ))
         return ( FALSE ) ;

   if (NULL == (hTTYWnd = InitInstance( hInstance, nCmdShow )))
      return ( FALSE ) ;

   while (GetMessage( &msg, NULL, 0, 0 ))
```

```
   {
      if (!TranslateAccelerator( hTTYWnd, ghAccel, &msg ))
      {
         TranslateMessage( &msg ) ;
         DispatchMessage( &msg ) ;
      }
   }

   if( ! UnmapViewOfFile( RangePtr ) ){
         return FALSE;
   }

   return ( (int) msg.wParam ) ;

} // end of WinMain()

//-----------------------------------------------------------------------
------
//  BOOL NEAR InitApplication( HANDLE hInstance )
//
//  Description:
//     First time initialization.  This registers information
//     such as window classes.
//
//  Parameters:
//     HANDLE hInstance
//        Handle to this instance of the application.
//
//-----------------------------------------------------------------

BOOL NEAR InitApplication( HANDLE hInstance )
{

   WNDCLASS  wndclass ;
   unsigned char p_cmd = 0x92;
                                                  /* 1001 0010 */
                                                  /* mode, Port A in,
Port B in,  Port C out -- 8255a */

   // register tty window class

   wndclass.style =         0 ;
   wndclass.lpfnWndProc =   TTYWndProc ;
   wndclass.cbClsExtra =    0 ;
   wndclass.cbWndExtra =    TTYEXTRABYTES ;
   wndclass.hInstance =     hInstance ;
   wndclass.hIcon =         LoadIcon( hInstance, MAKEINTRESOURCE(
TTYICON ) );
   wndclass.hCursor =       LoadCursor( NULL, IDC_ARROW ) ;
   wndclass.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1) ;
   wndclass.lpszMenuName =  MAKEINTRESOURCE( TTYMENU ) ;
   wndclass.lpszClassName = gszTTYClass ;

   return( RegisterClass( &wndclass ) ) ;

} // end of InitApplication()
```


Meta Exhibit 1011
Meta v. Mullen - Page 0053

```
//-----------------------------------------------------------------------
//  HWND NEAR InitInstance( HANDLE hInstance, int nCmdShow )
//
//  Description:
//     Initializes instance specific information.
//
//  Parameters:
//     HANDLE hInstance
//        Handle to instance
//
//     int nCmdShow
//        How do we show the window?
//


HWND NEAR InitInstance( HANDLE hInstance, int nCmdShow )
{
   HWND  hTTYWnd ;

   // load accelerators
   ghAccel = LoadAccelerators( hInstance, MAKEINTRESOURCE( TTYACCEL ) )
;

   // create the TTY window
   hTTYWnd = CreateWindow( gszTTYClass, gszAppName,
                           WS_OVERLAPPEDWINDOW,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           CW_USEDEFAULT, CW_USEDEFAULT,
                           NULL, NULL, hInstance, NULL ) ;

   if (NULL == hTTYWnd)
      return ( NULL ) ;

   ShowWindow( hTTYWnd, nCmdShow ) ;
   UpdateWindow( hTTYWnd ) ;

   fd_COM2 = fopen("record.dat","r+");

   return ( hTTYWnd ) ;

} // end of InitInstance()

//-----------------------------------------------------------------------
------
//  LRESULT FAR PASCAL TTYWndProc( HWND hWnd, UINT uMsg,
//                                 WPARAM wParam, LPARAM lParam )
//
//
HWND G_hWnd;
LRESULT FAR PASCAL TTYWndProc( HWND hWnd, UINT uMsg,
                               WPARAM wParam, LPARAM lParam )
{

   DWORD dwBlockSize=512;

   HDC          hdc;
   int loop, loop2;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0054

```
   COLORREF temp;
   int ix,iy, i, j;

   char tmpstr[81];
   char tmpstr2[81];
   char tmpstr3[81];
   POINT    *lppt;

   G_hWnd = hWnd;
   switch (uMsg)
   {

      case WM_CREATE:
         return ( CreateTTYInfo( hWnd ) ) ;

      case WM_COMMAND:
      {
         switch ( LOWORD( wParam ) )
         {
            case IDM_CONNECT:
               if (!OpenConnection( hWnd ))
                     {
                              MessageBox( hWnd, "Connection failed.",
gszAppName,
                              MB_ICONEXCLAMATION ) ;
                              sequence_errors = 0;
                              line_range_errors = 0;
                              col_range_errors = 0;
                     }
               break ;

            case IDM_DISCONNECT:
               KillTTYFocus( hWnd ) ;
               CloseConnection( hWnd ) ;
               break ;

                  case IDM_SAVE:
               if (idd_save())
                   MessageBox( hWnd, "idd_save failed.", gszAppName,
                              MB_ICONEXCLAMATION ) ;
                     else
                         MessageBox( hWnd, "idd_save succeeded.",
gszAppName,
                              MB_ICONEXCLAMATION ) ;
                     break;

            case IDM_SAVE_AZ_EL_RANGE:
               if (test_save_az_el_range())
                   MessageBox( hWnd, "test_save_az_el_range failed.",
gszAppName,
                              MB_ICONEXCLAMATION ) ;
                     //else
                     //    MessageBox( hWnd, "test_save_az_el_range
succeeded.", gszAppName,
               //                MB_ICONEXCLAMATION ) ;

               break ;
```

```
            case IDM_SETTINGS:
            {
               NPTTYINFO  npTTYInfo ;

               if (NULL == (npTTYInfo = GETNPTTYINFO( hWnd )))
                  return ( FALSE ) ;
               GoModalDialogBoxParam( GETHINST( hWnd ),
                                      MAKEINTRESOURCE( SETTINGSDLGBOX
), hWnd,
                                      (DLGPROC) SettingsDlgProc,
                                      (LPARAM) (LPSTR) npTTYInfo ) ;

               // if Connected, set new COM parameters

               if (CONNECTED( npTTYInfo ))
               {
                  if (!SetupConnection( hWnd ))
                     MessageBox( hWnd, "Settings failed!", gszAppName,
                                 MB_ICONEXCLAMATION ) ;
               }
            }
            break ;

            case IDM_SENDBLOCK:
                    hdc = GetDC(hWnd);

                        for (loop2 = 0; loop2 < NUM_ROWS; loop2++)
                        {
                              iy = loop2*2;
                              for ( loop = 0; loop < NUM_COLUMNS ;
loop++) {
                                    ix = loop*2;

                                    if ( VMO_Matrix[loop][loop2].status
!= -2)
                                    {
                                          temp =
((VMO_Matrix[loop][loop2].status + 1)*341) % 256 + 1;
                                          temp = PMap[temp];
                                    }
                                    else
                                          temp = 0;

                                    SetPixel(hdc,ix,iy,temp);
                                    SetPixel(hdc,ix+1,iy,temp);
                                    SetPixel(hdc,ix,iy+1,temp);
                                    SetPixel(hdc,ix+1,iy+1,temp);
                              }
                        }

               break;

            case IDM_SHOW_VBO_LIST:
                  {

                     hdc = GetDC(hWnd);
```

```
                        for (loop2 = 0; loop2 < total_VMOnum; loop2++)
                        {

                              temp = (loop2*341) % 256 + 1;
                              temp = PMap[temp];
                              for(i =
0;i<VMO_List[loop2].num_blades;i++)
                              {
                                    iy = 2 *
VMO_List[loop2].vertices[i].startrow;

                                    for (j =
VMO_List[loop2].vertices[i].startcolumn; j <=
VMO_List[loop2].vertices[i].endcolumn; j++)
                                    {
                                          ix = j*2;
                                          SetPixel(hdc,ix,iy,temp);
                                    }
                              }
                        }
                        lppt = (POINT *)malloc(sizeof(POINT)*100);
                        for (loop2 = 0; loop2 < total_VMOnum; loop2++)
                        {
                              lppt = (POINT *)realloc(lppt,
sizeof(POINT)*VMO_Polygons[loop2].num_indices + 2);
                              temp = (loop2*341) % 256 + 1;
                              temp = PMap[temp];

                              for(i = 0;i<
VMO_Polygons[loop2].num_indices;i++)
                              {
                                    lppt[i].x = 2 *
VMO_Polygons[loop2].vertices[i].x;
                                    lppt[i].y = 2 *
VMO_Polygons[loop2].vertices[i].y;

                              }
                              lppt[i].x = 2 *
VMO_Polygons[loop2].vertices[0].x;
                              lppt[i].y = 2 *
VMO_Polygons[loop2].vertices[0].y;
                              Polyline(hdc, lppt,
VMO_Polygons[loop2].num_indices +1);
                        }
                  }
            break;

            case IDM_ABOUT:
               GoModalDialogBoxParam ( GETHINST( hWnd ),
                                       MAKEINTRESOURCE( ABOUTDLGBOX ),
                                       hWnd,
                                       (DLGPROC) AboutDlgProc, 0L ) ;
               break;


            case IDM_FILE:
```


Meta Exhibit 1011
Meta v. Mullen - Page 0057

```
                if (test_process_file())
                    MessageBox( hWnd, "test_process_file failed.",
gszAppName,
                                MB_ICONEXCLAMATION ) ;
                break ;
            case IDM_GENERATE:
                if (test_generate_file())
                    MessageBox( hWnd, "test_generate_file failed.",
gszAppName,
                                MB_ICONEXCLAMATION ) ;

                break ;

            case IDM_MAKE_BLADES:
                if (make_vbo_list(hWnd))
                    MessageBox( hWnd, "Failure", gszAppName,
                                MB_ICONEXCLAMATION ) ;
                      else{
                            gcvt( duration, 7, buffer );
                            MessageBox( hWnd, buffer, gszAppName,
                                  MB_ICONEXCLAMATION ) ;
                      }
                break ;

            case IDM_RANGESET:
                GoModalDialogBoxParam ( GETHINST( hWnd ),
                                        MAKEINTRESOURCE( RANGESETDLGBOX
),
                                        hWnd,
                                        (DLGPROC) RangeSetDlgProc, 0L )
;

                break;

                 case IDM_OFFSETSET:
            {
                NPTTYINFO  npTTYInfo ;

                if (NULL == (npTTYInfo = GETNPTTYINFO( hWnd )))
                   return ( FALSE ) ;
                         GoDialogBoxParam( GETHINST( hWnd ),
                                       MAKEINTRESOURCE( IDD_OFFSETDLG ),
hWnd,
                                       (DLGPROC) OffsetDlgProc,
                                       (LPARAM) (LPSTR) npTTYInfo ) ;

            }
            break ;


            case IDM_EXIT:
                KillTTYFocus( hWnd ) ;
                CloseConnection( hWnd ) ;
                PostMessage( hWnd, WM_CLOSE, 0, 0L ) ;
                break ;
        }
    }
```

```
        break ;

        case WM_PAINT:
           PaintTTY( hWnd ) ;
           break ;

        case WM_SIZE:
           SizeTTY( hWnd, HIWORD( lParam ), LOWORD( lParam ) ) ;
           break ;

        case WM_HSCROLL:
           ScrollTTYHorz( hWnd, LOWORD( wParam ), HIWORD( wParam ) ) ;
           break ;

        case WM_VSCROLL:
           ScrollTTYVert( hWnd, LOWORD( wParam ), HIWORD( wParam ) ) ;
           break ;

        case WM_CHAR:
           ProcessTTYCharacter( hWnd, LOBYTE( LOWORD( wParam ) ) ) ;
           break ;

        case WM_LBUTTONDOWN:
             hdc = GetDC( hWnd );
               ix = LOWORD(lParam)/2;
               iy = HIWORD( lParam)/2;
               if(((ix>=0)&&(ix<NUM_COLUMNS)&&(iy>=0)&&(iy<NUM_ROWS))){
                     temp= (unsigned long)VMO_Matrix[ix][iy].range;
                     if(Hwnd_Offset) {


                           wsprintf(tmpstr," %d",ix);
                  SetDlgItemText(Hwnd_Offset,IDD_XCOORD, tmpstr);
                           wsprintf(tmpstr," %d",iy);
                  SetDlgItemText(Hwnd_Offset,IDD_YCOORD, tmpstr);
                           wsprintf(tmpstr," %d",temp);
                  SetDlgItemText(Hwnd_Offset,IDD_RANGE, tmpstr);
                           wsprintf(tmpstr," %d",total_VMOnum);
                  SetDlgItemText(Hwnd_Offset,IDD_TOTAL_VMO, tmpstr);
                           if(VMO_Matrix[ix][iy].status >=0)
                           {
                                 wsprintf(tmpstr,"
%d",VMO_List[VMO_Matrix[ix][iy].status].num_blades);
                           SetDlgItemText(Hwnd_Offset,IDC_VBO_NUMBLADES,
tmpstr);
                                wsprintf(tmpstr,"
%d",VMO_Polygons[VMO_Matrix[ix][iy].status].num_indices);
                     SetDlgItemText(Hwnd_Offset,IDC_VMO_NUMINDICES,
tmpstr);
                           }
                              else
                           {

SetDlgItemText(Hwnd_Offset,IDC_VBO_NUMBLADES, "No Active VMO");
                                SetDlgItemText(Hwnd_Offset,IDC_VMO_NUMINDICES,
"No Active VMO");
                           }
```

```
                        temp = VMO_Matrix[ix][iy].status;

                        wsprintf( tmpstr3,"%d",temp);

                SetDlgItemText(Hwnd_Offset,IDC_VBO_NUM,  tmpstr);
                SetDlgItemText(Hwnd_Offset,IDC_VBO_RANGE,tmpstr2);
                SetDlgItemText(Hwnd_Offset,IDC_VMO_NUM, tmpstr3);
                    }
            }

         ReleaseDC( hWnd, hdc );
         break;


        case WM_SETFOCUS:
         SetTTYFocus( hWnd ) ;
         break ;

      case WM_KILLFOCUS:
         KillTTYFocus( hWnd ) ;
         break ;

      case WM_DESTROY:
         DestroyTTYInfo( hWnd ) ;
         PostQuitMessage( 0 ) ;
         break ;

      case WM_CLOSE:
         if (IDOK != MessageBox( hWnd, "OK to close window?", "VICTOR
Control TTY",
                                 MB_ICONQUESTION | MB_OKCANCEL ))
            break ;

         // fall through

      default:
         return( DefWindowProc( hWnd, uMsg, wParam, lParam ) ) ;
   }
   return 0L ;

} // end of TTYWndProc()

//BOOL NEAR OpenConnection( HWND hWnd ) // Omitted, standard software
//BOOL NEAR SetupConnection( HWND hWnd ) // Omitted, standard software
//BOOL NEAR CloseConnection( HWND hWnd ) // Omitted, standard software
//int NEAR ReadCommBlock( HWND hWnd, LPSTR lpszBlock, int nMaxLength )
// Omitted, standard software

//BOOL NEAR WriteCommBlock( HWND hWnd, LPSTR lpByte , DWORD
dwBytesToWrite) // Omitted, standard software
//  BOOL NEAR WriteTTYBlock( HWND hWnd, LPSTR lpBlock, int nLength )
// Omitted, standard software

//  BOOL FAR PASCAL RangeSetDlgProc( HWND hDlg, UINT uMsg,
//                                   WPARAM wParam, LPARAM lParam )
//
```


Meta Exhibit 1011
Meta v. Mullen - Page 0060

```
//  Description:
//     This handles all of the user preference settings for
//     range min and max.
//
//  Parameters:
//     same as all dialog procedures
//
//-----------------------------------------------------------------
------
BOOL FAR PASCAL RangeSetDlgProc(HWND hdlg,UINT messg,
                            WPARAM wParam,LPARAM lParam)
{
      char intext[80];
switch (messg)
  {
  case WM_INITDIALOG:
      sprintf(intext, "%5.1f", Min_Range);
      SetDlgItemText(hdlg,IDD_MINRANGE,intext);

      sprintf(intext, "%5.1f", Max_Range);
      SetDlgItemText(hdlg,IDD_MAXRANGE,intext);
    return FALSE;
  case WM_COMMAND:
    switch (wParam)
    {
      case IDOK:
       GetDlgItemText(hdlg,IDD_MAXRANGE,intext,79);
         Max_Range = atof(intext);
       GetDlgItemText(hdlg,IDD_MINRANGE,intext,79);
         Min_Range = atof(intext);
            EndDialog(hdlg,TRUE);
            break;
      case IDCANCEL:
        EndDialog(hdlg,FALSE);
        break;
      default:
        return FALSE;
    }
    break;
  default:
    return FALSE;
  }
return TRUE;
}
//  DWORD FAR PASCAL CommWatchProc( LPSTR lpData )
//
//  Description:
//     A secondary thread that will watch for COMM events.
//
//  Parameters:
//     LPSTR lpData
//        32-bit pointer argument

int    min_nLength = 100;            // bc debug
int    max_nLength = 0;              // bc debug
NPTTYINFO   G_npTTYInfo;               // bc debug - used to be local
var in commwatchproc
```

```
//DWORD FAR PASCAL CommWatchProc( LPSTR lpData )
// Omitted, standard software

//  int Process_chars(char *abIn, int nLength)
//
//  Description:
//     parser for the laser data transfer protocol.
//
//  Parameters:
//     input string pointer
//     length - num chars.
//
//*********************************************************************
***
char history[1000];
int hist_index = 0;

int min_line = 1;
int max_line = 97;
int min_col = 0;
int max_col = 266;
char outstring[100];
int Process_chars(char *abIn, int nLength)

{
      int ii;
      static int line = 0;
      static int col = 0;
      static int * rowptr;
      unsigned char c;
      static int direction = UNKNOWN;
      static int state = UNKNOWN;
      static double scale_factor = {0.1};
      static long int num = {0};

      static int chars_remaining = 0;
      static int dec;
      int status = 0;

      for (ii = 0 ; ii < nLength ; ii++)
      {
            c = *(abIn+ii);

            switch( state )
            {
            case UNKNOWN:
                  if( c & 0x80 ) {
                     state = NUMERIC;
                     num = (c & 0x7f) << 7;
                  }
                  else if ( c == 'B')
                        state = STATE_B;
                  else if ( c == 'S')
                        state = STATE_S;
                  else if ( c == 'E')
                        state = STATE_E;
```

```
                else if ( c == 'F')
                {
                        state = UNKNOWN;
                        direction = UNKNOWN;
                }
                else if ( c == 'L')
                {
                        state = STATE_L;
                        chars_remaining = 3;
                    line = 0;
                }
                else if (isspace(c))
                        state = UNKNOWN;
                else{
                        state = UNKNOWN;
                        sequence_errors++;
                        status = 1;
                }
                break;

        case NUMERIC:
                if( c & 0x80 ) {
                   state = NUMERIC;
                   num = (c & 0x7f) << 7;
                   sequence_errors++;
                   status = 2;
                }
                else {
                        num |= c;
            //RangePtr->RngFrame[((line-1)*NUM_COLUMNS)+col] = num;
                        if( ((unsigned long)rowptr >= (unsigned
long)RangePtr->RngFrame) &&
                                rowptr < &(RangePtr->RngFrame[ NUM_ROWS *
NUM_COLUMNS ]))
                        {
                           *rowptr = num;
                        }
                        else
                                sequence_errors++;
                        if(ODD(line))
                                rowptr += direction;
                        else
                                rowptr -= direction;
                        state = UNKNOWN;
                }
                break;
        case STATE_B:
                if( c == 'F')
                        state = STATE_F;
                else{
                        state = UNKNOWN;
                        sequence_errors++;
                        status = 3;
                }
                break;
        case STATE_F:
                if(c == 'U')
```


Meta Exhibit 1011
Meta v. Mullen - Page 0063

```
                        direction = FORWARD;
                    else if(c == 'D')
                        direction = BACKWARD;
                    else{
                        direction = UNKNOWN;
                        sequence_errors++;
                        status = 4;
                    }
                  if ( G_npTTYInfo != NULL )
                    {
                        if (LOCALECHO( G_npTTYInfo ))
                        {
                            WriteTTYBlock( G_hWnd,
"Beginning_of_Frame Detected.\n\r", 30 ) ;

                        }
                    }
                    state = UNKNOWN;
                    break;
              case STATE_S:
                    if( c == 'L'){
                        state = STATE_L;
                        chars_remaining = 3;
                      line = 0;
                    } else if( c == 'C') {
                        state = STATE_C;
                        chars_remaining = 6;
                        scale_factor = 0;
                        dec = 0;
                    } else {
                        state = UNKNOWN;
                        sequence_errors++;
                        status = 5;
                    }
                    break;
              case STATE_L:
                    if( isdigit(c) ){
                        line = line*10 + (c - '0');
                        chars_remaining--;
                        if(chars_remaining == 0){
                            if( direction == FORWARD && EVEN(line)
                                || direction == BACKWARD &&
ODD(line) ) {
                                //col = NUM_COLUMNS - 1;
                                rowptr = &(RangePtr-
>RngFrame[((line-1)*NUM_COLUMNS)+(NUM_COLUMNS - 1)]);
                            } else if(direction != UNKNOWN) { /*
direction is FORWARD */
                                //col = 0;
                                rowptr = &(RangePtr-
>RngFrame[((line-1)*NUM_COLUMNS)]);
                            }
                            state = UNKNOWN;
                        }
                    } else {
                        state = UNKNOWN;
                        sequence_errors++;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0064

```
                        status = 6;
                    }
                    break;
            case STATE_C:
                    if( isdigit(c) ){
                        scale_factor = scale_factor * 10 + (c - '0');
                        chars_remaining--;
                        if( dec )
                            dec++;
                        if(chars_remaining == 0){
                            RangePtr->scale_factor = scale_factor /
pow(10.0,dec-1);
                            if( RangePtr->scale_factor == 0.0 )
                                RangePtr->scale_factor = 1.0;
                            state = UNKNOWN;
                        }
                    } else if (c == '=') {
                        chars_remaining--;
                    } else if (c == '.') {
                        chars_remaining--;
                        dec = 1;
                    } else {
                        state = UNKNOWN;
                        sequence_errors++;
                        status = 7;
                    }
                    break;
            case STATE_E:
                    if( c != 'F')
                    {
                        sequence_errors++;
                        status = 8;
                    }
                    if ( G_npTTYInfo != NULL )
                        if (LOCALECHO( G_npTTYInfo ))
                        {
                            sprintf(
                                outstring,
                                "End_of_Frame Detected. %d SeqErrs;
%d LinRanErrs; %d ColRanErrs.\n\r",
                                sequence_errors, line_range_errors,
col_range_errors);
                            WriteTTYBlock( G_hWnd, outstring,
strlen(outstring) ) ;

                        }
                    state = UNKNOWN;
                    direction = UNKNOWN;
                    //hist_index = 0;
                    break;
            }
            if(status)
                    break;
        }

        return status;
}
```


Meta Exhibit 1011
Meta v. Mullen - Page 0065

```
#define MEAN_X (((j - blade_ptr->start_x) + 1)/2)
#define MAX_DELTA_RANGE 1.1

int old_make_vbo_list( void )
{
      double      Max_Slope                     = 0.25;
      int         status                              = 0;
      int         blade_in_progress       = FALSE;
      double      current_range_sample    = 0;
      double      last_range_sample       = 0;
      double      delta_range                   = 0;
      int * sample_ptr;
      Vbo *   blade_ptr;
      int     i,j;

      blade_ptr = &VBO_List[0];
      num_blades = 0;
      sample_ptr = &RangePtr->RngFrame[0];
      for( i = 0; i < NUM_ROWS; i++) // process each row of the scanned
in image
      {
            blade_in_progress = FALSE;
            for (j = 0; j < NUM_COLUMNS; j++)
            {
                  current_range_sample = *sample_ptr * RangePtr-
>scale_factor;
                  if( blade_in_progress )
                  {
                        if( (current_range_sample >= Min_Range) &&
(current_range_sample <= Max_Range) )
                        {
                              // test for inclusion in current blade
                              delta_range = fabs(current_range_sample -
last_range_sample);

                              if( delta_range > MAX_DELTA_RANGE )
                              {
                                    // end current blade
                                    blade_ptr->end_x = j-1;
                                    blade_ptr->end_z =
last_range_sample;
                                    blade_ptr->mean_range /=
(blade_ptr->end_x - blade_ptr->start_x) + 1;
                                    blade_ptr++;
                                    num_blades++;
                                    // start new blade
                                    blade_ptr->blade_num = num_blades;
                                    blade_ptr->row_index = i;
                                    blade_ptr->start_x = j;
                                    blade_ptr->start_z =
current_range_sample;
                                    blade_ptr->mean_range =
current_range_sample;
                              }
```


Meta Exhibit 1011
Meta v. Mullen - Page 0066

```
                                else
                                {
                                        // add current point to current
blade, ...
                                        blade_ptr->mean_range +=
current_range_sample;
                                }
                        }
                        else
                        {
                                // end current blade
                                blade_ptr->end_x = j-1;
                                blade_ptr->end_z = last_range_sample;
                                blade_ptr->mean_range /= (blade_ptr-
>end_x - blade_ptr->start_x) + 1;
                                blade_ptr++;
                                num_blades++;

                                blade_in_progress = FALSE;
                        }
                }
                else
                {
                        if( (current_range_sample >= Min_Range) &&
(current_range_sample <= Max_Range) )
                        {
                                // start new blade
                                blade_ptr->blade_num = num_blades;
                                blade_ptr->row_index = i;
                                blade_ptr->start_x = j;
                                blade_ptr->start_z =
current_range_sample;
                                blade_ptr->mean_range =
current_range_sample;

                                blade_in_progress = TRUE;
                        }
                }
                sample_ptr++;                   // advance pointer to
point to next range sample
                last_range_sample = current_range_sample;
            } // end column loop
        }// end row loop
        return status;
}
Vbo * FindBlade( int ix, int iy)
{
        int i = 0;
      while( (VBO_List[i].row_index < iy ) &&
                (i < num_blades) )
            i++;
        while( (VBO_List[i].end_x < ix) &&
                (VBO_List[i].row_index == iy) &&
                (i < num_blades) )
            i++;
        if( ( VBO_List[i].start_x <= ix ) &&
            ( VBO_List[i].end_x >= ix ) &&
```

```
            ( VBO_List[i].row_index == iy ) &&
            ( i < num_blades ) )
                        return &VBO_List[i];
      else
            return NULL;

}


#define LO_BYTE(x) (x & 0x7f)
#define HI_BYTE(x) (((x & 0x3f80) >> 7) | 0x80)

#define RANGE(i,j) (RangePtr->RngFrame[i*NUM_COLUMNS + j] * RangePtr-
>scale_factor)
int idd_save( void )
{
      FILE *      file_ptr;
    char *  filename = "savedata.raw";
      int         line, col, range;
      int     max_range = 1600;
      int     min_range = 0;
      char    k,l;
      double  range_step;

      file_ptr = fopen(filename,"wb");
      if(! file_ptr )
            return 1;
    fprintf(file_ptr,"EFBFUSC=1.000");
      range_step = (double)(max_range - min_range) /
(double)NUM_COLUMNS;
      for( line = 1; line<NUM_ROWS+1; line++){
            fprintf(file_ptr,"\nSL%03d",line);
            // write one line to the file

            for(col = 0; col<NUM_COLUMNS; col++){
            if( EVEN(line) )
                        range=(int)RANGE(line,(NUM_COLUMNS-1)- col);
                  else
                        range=(int)RANGE(line,col);
                  l = LO_BYTE(range);
                  k = HI_BYTE(range);
                  fprintf(file_ptr,"%c%c",k,l);
            }
    }
    fprintf(file_ptr,"EF");
    fclose( file_ptr );
      return 0;

}

#define MYSTERY -1
#define OUTOFRANGE -2
int make_vbo_list( HWND hWnd )
{

      int * sample_ptr;
      int     i,j,VMOnum;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0068

```
	VMOnum = 0;
	old_make_vbo_list();
	sample_ptr = &RangePtr->RngFrame[0];
	for (i = 0; i < NUM_ROWS; i++)

	{
		for( j = 0; j < NUM_COLUMNS; j++) // process each row of
the scanned in image
		{
			VMO_Matrix_Temp[i][j].range = (float)(*sample_ptr *
RangePtr->scale_factor);
			if(inrange(VMO_Matrix_Temp[i][j].range))
				VMO_Matrix_Temp[i][j].status = MYSTERY;
			else
				VMO_Matrix_Temp[i][j].status = OUTOFRANGE;
			sample_ptr++;                // advance pointer to
point to next range sample

		} // end column loop
	}// end row loop
	//Switch to a standard cartesian system
	for (i = 0; i < NUM_COLUMNS; i++)

	{
		for( j = 0; j < NUM_ROWS; j++) // process each row of the
scanned in image
		{
			VMO_Matrix[i][j] = VMO_Matrix_Temp[j][i];
		}
	}
	start = clock();//start timer


	for( i = 0; i < NUM_COLUMNS; i++) //iterate through each cell,
lots of room for
                                                            //improvement
here - implement logic and put away the shotgun!!
	{
		for (j = 0; j < NUM_ROWS; j++)
		{
			is_started = 0;//used to keep a tab on depth of the
current VMO
			num_cells = 0;//used to keep track of how large VMO's
are
			if(VMO_Matrix[i][j].status == MYSTERY)//Unaccounted
for cell.
			{
				FindVMO(i, j, 1, i, i, MYSTERY, VMOnum);//get
the VMO

				if(num_cells > MIN_VMOSIZE)//Only make a new
avatar if it is worth showing
				{
					//VMOnum+=OpenVMO(VMOnum);//adjust VMO's
to contain only one VBO per horizontal line
					VMOnum++;
```

```
                        }
                        else//otherwise go back and mark the cells as
OUTOFRANGE (they will not be
                              //part of a different VMO)
                        {
                              FindVMO(i, j, 1, i, i, VMOnum,
OUTOFRANGE);
                        }

                  }
            }
      }
      total_VMOnum = VMOnum;
      if(make_blades_to_objects(hWnd))
      {
            finish = clock();//end timer and convert to seconds
            duration = (double)(finish - start) / CLOCKS_PER_SEC;
            return 1;
      }
      finish = clock();//end timer and convert to seconds
      duration = (double)(finish - start) / CLOCKS_PER_SEC;
      //printstatusmatrix();//prints the status matrix to
'StatusMat.txt'-huge file!
      return 0;//status - not really used right now;
}
int OpenVMO(int VMOnum)
{
      int j, num_newguys;
      int x, t;
      num_newguys = 0;
      for(j = 0;j < NUM_ROWS;j++)
      {

            while(num_bladesinline(j, VMOnum)>1)
            {
                  x = -1;
                  while((++x < NUM_COLUMNS)&&(VMO_Matrix[x][j].status
!= VMOnum));
                  if(x < NUM_COLUMNS)
                  {

                        num_cells2 = 0;
                        t = VMOnum+num_newguys+1;
                        SeperateVMO(x, j, 1, x, x,VMOnum, t);
                        if (num_cells2 > 2)//Only make a new avatar if
it is worth showing
                        {
                              num_newguys++;
                        }
                        else//otherwise go back and mark the cells as
OUTOFRANGE (they will not be
                              //part of a different VMO)
                        {
                              SeperateVMO(x, j, 1, x, x,t, OUTOFRANGE);
                        }
                        while((++x <
NUM_COLUMNS)&&(VMO_Matrix[x][j].status == t));
```


Meta Exhibit 1011
Meta v. Mullen - Page 0070

```
                }
            }
        }
        return num_newguys;
}
int inrange(float rangedata)
{
        if((rangedata >= Min_Range) && (rangedata <= Max_Range))
            return 1;

        return 0;
}
int FindVMO(int X_seed, int Y_seed, char Direction, int prev_XL, int
prev_XR, int seedval, int fillval)
{
        int x, y, xl, xr;
        y = Y_seed;
        xl = X_seed;
        xr = X_seed;
        if(!is_started)//keep max VMO depth as MAXVMO_DEPTH
        {
            initial_range = (double)getRange(xl, y);
            is_started = 1;
        }

            xl = ScanL(xl, y, seedval, fillval);//find left & right
side of VBO (if any)
            xr = ScanR(xr, y, seedval, fillval);

                num_cells+=((xr - xl) - 1);//keep tabs on the number
of cells in the current VMO.


        for(x = xl;x<=xr;x++)//Check for connection in the next
row(direction)
        {
            if((VMO_Matrix[x][y + Direction].status ==
seedval)&&(isZmatch(x, y, x, y+ Direction)))
            {

                    x = FindVMO(x, y + Direction, Direction, xl,
xr, seedval, fillval);
            }

        }

        for (x = xl;x < prev_XL;x++)//Check for connection in the prev
row( - direction)
        {
            if((VMO_Matrix[x][y - Direction].status ==
seedval)&&(isZmatch(x, y, x, y - Direction)))
            {

                    x = FindVMO(x, y - Direction, Direction, xl,
xr, seedval, fillval);
            }
```


Meta Exhibit 1011
Meta v. Mullen - Page 0071

```
	}
	for (x = prev_XR;x <= xr;x++)//Check for connection in the prev
row( - direction)
	{
		if((VMO_Matrix[x][y - Direction].status ==
seedval)&&(isZmatch(x, y, x, y - Direction)))
		{

				x = FindVMO(x, y - Direction, (char)(-
Direction), xl, xr, seedval, fillval);
		}
	}
	return xr;
}
/*
	This function is designed to remove Those VBO's that make a VMO
contain multiple VBO's in the same row.  It works by changing cells in
adjoining rows in the down direction only
*/
int SeperateVMO(int X_seed, int Y_seed, char Direction,int prev_XL, int
prev_XR, int seedval, int fillval)
{
	int x, y, xl, xr;
	y = Y_seed;
	xl = X_seed;
	xr = X_seed;

	xl = ScanL(xl, y, seedval, fillval);
	xr = ScanR(xr, y, seedval, fillval);
	num_cells2+=((xr - xl) - 1);//keep tabs on the number of cells in
the current VMO.


	for(x = xl;x<=xr;x++)
	{
		if((VMO_Matrix[x][y + Direction].status ==
seedval)&&(isZmatch(x, y, x, y+ Direction)))
		{

				x = SeperateVMO(x, y + Direction, Direction,
xl, xr, seedval, fillval);
		}

	}
	for (x = xl;x < prev_XL;x++)//Check for connection in the prev
row( - direction)
	{
		if((VMO_Matrix[x][y + Direction].status ==
seedval)&&(isZmatch(x, y, x, y + Direction)))
		{

				x = SeperateVMO(x, y + Direction, Direction,
xl, xr, seedval, fillval);
		}

	}
```

```
	for (x = prev_XR;x <= xr;x++)//Check for connection in the prev
row( - direction)
	{
		if((VMO_Matrix[x][y + Direction].status ==
seedval)&&(isZmatch(x, y, x, y + Direction)))
		{

				x = SeperateVMO(x, y + Direction, Direction,
xl, xr, seedval, fillval);
		}
	}
	return xr;
}
int ScanR(int xr, int y, int seedval, int fillval)
{
	int x,isblade;
	x = xr;
	isblade = 1;

		while((++x < NUM_COLUMNS)&&(isblade))
		{

			if((VMO_Matrix[x][y].status == seedval)&&(isZmatch(x
-1, y, x, y)))
				VMO_Matrix[x][y].status = fillval;
			else
				isblade = 0;

		};
	//}
	return --x;//inclusive return - last x bad
}
int ScanL(int xl, int y, int seedval, int fillval)
{
	int x,isblade;
	x = xl;
	isblade = 1;
	if(VMO_Matrix[x][y].status == seedval)
	{
		VMO_Matrix[x][y].status = fillval;
		while((--x >= 0)&&(isblade))
		{

			if((VMO_Matrix[x][y].status == seedval)&&(isZmatch(x,
y, x + 1, y)))
				VMO_Matrix[x][y].status = fillval;
			else
				isblade = 0;

		};
	}
	return ++x;//inclusive return - last x bad
}
int isZmatch(int x1, int y1, int x2, int y2)
{
	float delta_range;
	//if out of range in the x or y direction return a miss
```


Meta Exhibit 1011
Meta v. Mullen - Page 0073

```
	if((x1 < 0)|(x2 < 0)|(x1 >= NUM_COLUMNS)|(x2 >= NUM_COLUMNS))
		return 0;
	if((y1 < 0)|(y2 < 0)|(y1 >= NUM_ROWS)|(y2 >= NUM_ROWS))
			return 0;
	//If depth of current VMO in range
	if((fabs(initial_range - (double)VMO_Matrix[x1][y1].range) >
MAXVMO_DEPTH) ||(fabs(initial_range - (double)VMO_Matrix[x2][y2].range)
> MAXVMO_DEPTH))
			return 0;
	delta_range = (float)fabs((double)VMO_Matrix[x1][y1].range -
(double)VMO_Matrix[x2][y2].range);
	if(delta_range < MAX_DELTA_RANGE)
			return 1;
	return 0;
}
float getRange(int x, int y)
{
	return (VMO_Matrix[x][y].range);
}
int lineavailable(int y, int fillval)
{
	int x, ret_val;
	ret_val = 1;
	x = -1;
	while(++x < NUM_COLUMNS)
	{
		if(VMO_Matrix[x][y].status == fillval)
		{
			ret_val = 0;
			break;
		}
	}
	return ret_val;
}
//Determines the number of VBO's contained in a row
int num_bladesinline(int y, int fillval)
{
	int x, ret_val;
	ret_val = 0;
	x = -1;
	while(++x < NUM_COLUMNS)
	{
		if(VMO_Matrix[x][y].status == fillval)//if a VBO
		{
			ret_val++;//one more VBO
			//while current VBO
			while((++x < NUM_COLUMNS)&&(VMO_Matrix[x][y].status
== fillval));
			//--x;//is reset to a miss in loop control

		}
	}
	return ret_val;
}
int make_blades_to_objects(HWND hWnd)
{
	int i,  row, vertexnum, start, end;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0074

```
	if((VMO_List = (struct VMO_Struct *)malloc(sizeof(struct
VMO_Struct) * total_VMOnum))==NULL)
	return 1;
	for(i = 0;i < total_VMOnum;i++)
	{
		row = -1;
		while(!num_bladesinline(++row, i));
		if((VMO_List[i].vertices = (struct V_Point
*)malloc(sizeof(struct V_Point) * (NUM_ROWS - row + 1)))==NULL)
		return 2;
		--row;
		vertexnum = 0;
		while(num_bladesinline(++row, i))
		{
			start = find_bladestart(row, i);
			end = find_bladeend(row, i);
			VMO_List[i].vertices[vertexnum].startrange =
VMO_Matrix[start][row].range;
			VMO_List[i].vertices[vertexnum].endrange =
VMO_Matrix[end][row].range;
			VMO_List[i].vertices[vertexnum].startrow = row;
			VMO_List[i].vertices[vertexnum].endrow = row;
			VMO_List[i].vertices[vertexnum].endcolumn = end;
			VMO_List[i].vertices[vertexnum++].startcolumn =
start;
		}
		if((VMO_List[i].vertices = (struct V_Point *)realloc(( void
*)VMO_List[i].vertices, sizeof(struct V_Point) * (vertexnum +
1)))==NULL)
		return 3;
		VMO_List[i].num_blades = vertexnum;
	}
	polygonize_VMO(hWnd);
	//Load_Resource(hWnd);
	return (0);
}

int find_bladestart(int y, int fillval)
{
	int x = -1;
		while((++x < NUM_COLUMNS)&&(VMO_Matrix[x][y].status !=
fillval));
	return x;
}
int find_bladeend(int y, int fillval)
{
	int x = NUM_COLUMNS;
		while((--x >= 0)&&(VMO_Matrix[x][y].status != fillval));
	return x;

}
int polygonize_VMO(HWND hWnd)
{
	int i, current_vertex, num_set_vertices, last_set_vertex,
position;
	current_vertex = 0;
	position = 0;
```

```
	if((VMO_Polygons = (struct VMO_PolyStruct *)malloc(sizeof(struct
VMO_PolyStruct) * total_VMOnum))==NULL)
	return 4;
	for(i = 0;i < total_VMOnum;i++)
	{
		if((VMO_Polygons[i].vertices = (struct IFS_Point
*)malloc(sizeof(struct IFS_Point) * VMO_List[i].num_blades * 2 +
2))==NULL)
		return 5;
		last_set_vertex = 0;
		current_vertex = 2;
		num_set_vertices = 1;
		VMO_Polygons[i].vertices[0].x =
VMO_List[i].vertices[0].startcolumn;
		VMO_Polygons[i].vertices[0].y =
VMO_List[i].vertices[0].startrow;
		VMO_Polygons[i].vertices[0].z =
(int)VMO_List[i].vertices[0].startrange;
		//for(j = 2;j < VMO_List[i].num_blades;j++)
		while(current_vertex < VMO_List[i].num_blades)
		{
			if(!(polygon_iteration_down(i, last_set_vertex,
current_vertex)))//If successfully approximated
				current_vertex++;
			else//otherwise, set the previous vertex and begin
anew
			{
				VMO_Polygons[i].vertices[num_set_vertices].x =
VMO_List[i].vertices[current_vertex - 1].startcolumn;
				VMO_Polygons[i].vertices[num_set_vertices].y =
VMO_List[i].vertices[current_vertex - 1].startrow;
				VMO_Polygons[i].vertices[num_set_vertices].z =
(int)VMO_List[i].vertices[current_vertex - 1].startrange;
				last_set_vertex = current_vertex - 1;
				current_vertex++;
				num_set_vertices++;
			}
		}
		VMO_Polygons[i].vertices[num_set_vertices].x =
VMO_List[i].vertices[VMO_List[i].num_blades - 1].startcolumn;
		VMO_Polygons[i].vertices[num_set_vertices].y =
VMO_List[i].vertices[VMO_List[i].num_blades - 1].startrow;
		VMO_Polygons[i].vertices[num_set_vertices++].z =
(int)VMO_List[i].vertices[VMO_List[i].num_blades - 1].startrange;
		VMO_Polygons[i].vertices[num_set_vertices].x =
VMO_List[i].vertices[VMO_List[i].num_blades-1].endcolumn;
		VMO_Polygons[i].vertices[num_set_vertices].y =
VMO_List[i].vertices[VMO_List[i].num_blades-1].endrow;
		VMO_Polygons[i].vertices[num_set_vertices].z =
(int)VMO_List[i].vertices[VMO_List[i].num_blades-1].endrange;
		last_set_vertex = VMO_List[i].num_blades - 1;
		num_set_vertices++;
		current_vertex = VMO_List[i].num_blades - 3;
		while(current_vertex >=0)
		{
			if(!(polygon_iteration_up(i, last_set_vertex,
current_vertex)))//If successfully approximated
```


Meta Exhibit 1011
Meta v. Mullen - Page 0076

```
                        current_vertex--;
                    else//otherwise, set the previous vertex and begin
anew
                    {
                        VMO_Polygons[i].vertices[num_set_vertices].x =
VMO_List[i].vertices[current_vertex + 1].endcolumn;
                        VMO_Polygons[i].vertices[num_set_vertices].y =
VMO_List[i].vertices[current_vertex + 1].endrow;
                        VMO_Polygons[i].vertices[num_set_vertices].z =
(int)VMO_List[i].vertices[current_vertex + 1].endrange;
                        last_set_vertex = current_vertex + 1;
                        num_set_vertices++;
                        current_vertex--;
                    }
                }
                VMO_Polygons[i].vertices[num_set_vertices].x =
VMO_List[i].vertices[0].endcolumn;
                VMO_Polygons[i].vertices[num_set_vertices].y =
VMO_List[i].vertices[0].endrow;
                VMO_Polygons[i].vertices[num_set_vertices++].z =
(int)VMO_List[i].vertices[0].endrange;
                if((VMO_Polygons[i].vertices = (struct IFS_Point
*)realloc(VMO_Polygons[i].vertices, sizeof(struct IFS_Point) *
num_set_vertices + 1))==NULL)
                return 6;
                VMO_Polygons[i].num_indices = num_set_vertices;
        }
        if(Load_Resource(hWnd))
        {
                return 7;

        }
        else
        {
                Load_SharedMem();
        }
        return 0;
}
void Load_SharedMem(void)
{
        long *intwriter, i, j, position = 0;
        float *floatwriter, azimuth, elevation, depth, range;

        floatwriter = (float *)lpVictorMap;
        intwriter = (long *)lpVictorMap;
        intwriter[position++] = CURRENTLY_WRITING;
        intwriter[position++] = total_VMOnum;
        /*#define RADS_PER_PIXEL (float)(0.0095)
        #define EL(i) ((NUM_ROWS/2 - (i+1))*RADS_PER_PIXEL)
        #define AZ(j) (((j) - NUM_COLUMNS/2)*RADS_PER_PIXEL)
        #define RANGE(i,j) (RangePtr->RngFrame[i*NUM_COLUMNS + j] *
RangePtr->scale_factor)

        #define CAMHEIGHT (1.6)
        #define XCOORD(az,el,d) (sin(az) * cos(el) * (d))
        //note: VRML looks in the -Z direction ==> all Z coords are
negative
```


Meta Exhibit 1011
Meta v. Mullen - Page 0077

```
	#define ZCOORD(az,el,d) (cos(az) * cos(el) * (d))
	#define YCOORD(az,el,d,z) ((sin(el) * cos(az) * (d)) + CAMHEIGHT
- sin(RADS_PER_PIXEL) * (z))
	#define HALF_LASER_PIXEL (float)(RADS_PER_PIXEL/2.0)*/
	for (i = 0; i < total_VMOnum; i++)
	{
		intwriter[position++] = VMO_Polygons[i].num_indices;
		for(j = 0;j< VMO_Polygons[i].num_indices;j++)
		{
			azimuth = AZ(VMO_Polygons[i].vertices[j].x)-
HALF_LASER_PIXEL;//build logic to determine left or right side
			elevation =
EL(VMO_Polygons[i].vertices[j].y)+HALF_LASER_PIXEL;//build logic to
determine top or bottom
			depth = (float)VMO_Polygons[i].vertices[j].z;
			range = (float)ZCOORD(azimuth,elevation,depth);

			floatwriter[position++] =
(float)XCOORD(azimuth,elevation,depth);
			floatwriter[position++] =
(float)YCOORD(azimuth,elevation,depth,range);
			floatwriter[position++] = -range;//just so the scenes
are visible

		}

	}
	intwriter[0] = READY_TO_READ;
}
int polygon_iteration_down(int curr_VMO, int last_set_vertex, int
current_vertex)
{
	//z with respect to x
	int i, rise, run, is_ok;
	float predicted_column, slope, intercept;
    ///return 1;//debug

	if(VMO_List[curr_VMO].vertices[last_set_vertex].startrange ==
VMO_List[curr_VMO].vertices[current_vertex].startrange)
	{

	if(fabs(VMO_List[curr_VMO].vertices[last_set_vertex].startcolumn
- VMO_List[curr_VMO].vertices[current_vertex].startcolumn)>
ALLOWED_ITERATION_ERROR)
			return 1;
		else
			return 0;
	}
	else//otherwise, get the equation of the current line
	{
		rise =
VMO_List[curr_VMO].vertices[last_set_vertex].startcolumn -
VMO_List[curr_VMO].vertices[current_vertex].startcolumn;
		run =
(int)VMO_List[curr_VMO].vertices[last_set_vertex].startrange -
(int)VMO_List[curr_VMO].vertices[current_vertex].startrange;
		slope = (float)rise/(float)run;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0078

```
            intercept = slope *
VMO_List[curr_VMO].vertices[last_set_vertex].startrange -
VMO_List[curr_VMO].vertices[last_set_vertex].startcolumn;

      }
      is_ok = 1;
      i = last_set_vertex;
      i++;
      while((is_ok)&&(i <= current_vertex))
      {
            predicted_column = slope *
VMO_List[curr_VMO].vertices[i].startrange + intercept;
            if(fabs(predicted_column -
(float)VMO_List[curr_VMO].vertices[i].startcolumn) <
ALLOWED_ITERATION_ERROR)
                  is_ok = 0;
            else
            {
                  i++;
            }
      }
      return is_ok;
}
int polygon_iteration_up(int curr_VMO, int last_set_vertex, int
current_vertex)
{
      //z with respect to x
      int i, rise, run, is_ok;
      float predicted_column, slope, intercept;
    ///return 1;//debug

      if(VMO_List[curr_VMO].vertices[last_set_vertex].endrange ==
VMO_List[curr_VMO].vertices[current_vertex].endrange)
      {

      if(fabs(VMO_List[curr_VMO].vertices[last_set_vertex].endcolumn -
VMO_List[curr_VMO].vertices[current_vertex].endcolumn)>
ALLOWED_ITERATION_ERROR)
                  return 1;
            else
                  return 0;
      }
      else//otherwise, get the equation of the current line
      {
            rise =
VMO_List[curr_VMO].vertices[last_set_vertex].endcolumn -
VMO_List[curr_VMO].vertices[current_vertex].endcolumn;
            run =
(int)VMO_List[curr_VMO].vertices[last_set_vertex].endrange -
(int)VMO_List[curr_VMO].vertices[current_vertex].endrange;
            slope = (float)rise/(float)run;
            intercept = slope *
VMO_List[curr_VMO].vertices[last_set_vertex].endrange -
VMO_List[curr_VMO].vertices[last_set_vertex].endcolumn;

      }
      is_ok = 1;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0079

```
      i = last_set_vertex;
      i--;
      while((is_ok)&&(i >= current_vertex))
      {
            predicted_column = slope *
VMO_List[curr_VMO].vertices[i].endrange + intercept;
            if(fabs(predicted_column -
(float)VMO_List[curr_VMO].vertices[i].endcolumn) <
ALLOWED_ITERATION_ERROR)
                  is_ok = 0;
            else
            {
                  i--;
            }
      }
      return is_ok;

}

int Load_Resource(HWND hWnd)
{
      long ret_val = 100;
      hVictorMap = GetMapHandle(SharedMapName, MAPSIZE);

    if (hVictorMap == NULL)//Alredy exists, or unable to create
      {
        hVictorMap =  OpenFileMapping(FILE_MAP_READ, TRUE,
SharedMapName);
            if(hVictorMap)
            {
                        lpVictorMapRead = (void *)MapViewOfFile(
                        hVictorMap,       /* handle of object to map
view of    */
                        FILE_MAP_WRITE , /* read/write access        */
                        0,              /* high offset:   map from  */
                        0,              /* low offset:    beginning */
                        0);             /* default: map entire file */
                  if (lpVictorMapRead)
                  {
                        ret_val = 0;
                  }
                  else
                  {
                        ret_val = 1;
                  }
            }
            else
            {
                        ret_val = 2;

            }
      }
      else
      {
        lpVictorMap = (void *)MapViewOfFile(
        hVictorMap,       /* handle of object to map view of    */
        FILE_MAP_WRITE, /* read/write access        */
```

```
        0,              /* high offset:   map from  */
        0,              /* low offset:    beginning */
        0);             /* default: map entire file */
            if (lpVictorMap)
            {
                  ret_val = 0;
            }
            else
            {
                  ret_val = 4;
            }
      }
      return ret_val;
}

HANDLE GetMapHandle(char *MapName, long mapsize){
      HANDLE rethandle;
            rethandle = CreateFileMapping(
        (HANDLE) 0xFFFFFFFF, /* use paging file    */
        NULL,                /* no security attr.  */
        PAGE_READWRITE,      /* read/write access  */
        0,                   /* size: high 32-bits */
        mapsize,  /* size: low 32-bits  */
        MapName);   /* name of map object */
      if (rethandle != NULL && GetLastError() == ERROR_ALREADY_EXISTS)
{
            CloseHandle(rethandle);
            rethandle = NULL;
      }
      return rethandle;
}

//-----------------------------------------------------------------------
------
//  End of File: vmo_gen.c
```


Meta Exhibit 1011
Meta v. Mullen - Page 0081

2 C-Java Interface Code

```
#include <native.h>
/* Header for class CVMOObject */
#ifndef _Included_CVMOObject
#define _Included_CVMOObject
#pragma pack(4)

typedef struct ClassCVMOObject {
    char PAD;     /* ANSI C requires structures to have a least one
member */
} ClassCVMOObject;
HandleTo(CVMOObject);

#pragma pack()
#ifdef __cplusplus
extern "C" {
#endif
struct Hfloatarrayobject;
struct Hintarrayobject;
extern long CVMOObject_loadVMOObjectinC(struct HCVMOObject *,struct
Hfloatarrayobject *,struct Hintarrayobject *);
extern long CVMOObject_getNuminNextVMOObjectinC(struct HCVMOObject *);
extern long CVMOObject_getNumofNextVMOObjectinC(struct HCVMOObject *);
extern long CVMOObject_initVMOObjectinC(struct HCVMOObject *);
extern long CVMOObject_destroyVMOObjectinC(struct HCVMOObject *);
#ifdef __cplusplus
}
#endif
#endif

#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <windows.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define FALSE                 0
#define TRUE                  !(FALSE)
#define PIx2      6.2830
typedef struct point3d{
      float x;
      float y;
      float z;
} ;
typedef struct VRMLObject{
            long num_vertices;
            struct point3d *pointz;
            long *indices;
} ;

long current_object;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0082

```
long num_objects;
long isinitialized = FALSE;
struct VRMLObject *IFSarray;

long CVMOObject_loadVMOObjectinC(struct HCVMOObject *,struct
Hfloatarrayobject *,struct Hintarrayobject *);
long CVMOObject_getNuminNextVMOObjectinC(struct HCVMOObject *);
long CVMOObject_initVMOObjectinC(struct HCVMOObject *);
long CVMOObject_destroyVMOObjectinC(struct HCVMOObject *);
long CVMOObject_getNuminNextVMOObjectinC(struct HCVMOObject *);
long generateISF(void);


/*
      This initializes the whole thing.
*/

long CVMOObject_initVMOObjectinC(struct HCVMOObject *javathis){
      if(isinitialized == TRUE)//allow only one at a time
            return -2;
      //Need to communicate with shared resource, ret that
            current_object = 0;
            num_objects = 10;
            //generateISF();
            if(generateISF()<0)
                  return -1;
      isinitialized = TRUE;
      return num_objects;
}
/*
      This temporary function merely gives some avatars to insert into
VRML.
*/
long generateISF(void){
      int i,j, scaler;
      float angle;
      if(!(IFSarray = (struct VRMLObject *) GlobalAlloc(GMEM_FIXED,
num_objects * sizeof(struct VRMLObject))))
            return -1;
      scaler = 1;


      for(i = 0;i<num_objects;i++){

            IFSarray[i].num_vertices = i + 3;

            if('(IFSarray[i].pointz = (struct point3d
*)GlobalAlloc(GMEM_FIXED,(num_objects + 1) * sizeof(struct point3d))))
                  return -1;

            if(!(IFSarray[i].indices = (long
*)GlobalAlloc(GMEM_FIXED,(num_objects + 3) * sizeof(long))))
                  return -1;

            j = 0;
            for(angle = 0;angle <= PIx2;angle += (float)(PIx2/(i +
3))){
```


Meta Exhibit 1011
Meta v. Mullen - Page 0083

```
                    IFSarray[i].pointz[j].z = (float)-10*scaler;
                    IFSarray[i].pointz[j].y = (float)-10*scaler *
(float)sin(angle);
                    IFSarray[i].pointz[j].x = (float)-10*scaler *
(float)cos(angle);
                    IFSarray[i].indices[j] = j;
                    j++;
              }
              scaler = scaler + 5;
              IFSarray[i].indices[j] = j;
              IFSarray[i].indices[j+1] = -1;
        }
        return 1;
}
/*
        This function frees all of the available memory associated with
the
current array of vertices.
*/
long CVMOObject_destroyVMOObjectinC(struct HCVMOObject *CVMOObjectthis)
{
        int i, j;

        /*if(isinitialized == TRUE)
        {
              for(i = 0;i<num_objects;i++){
                    for(j = 0;j<IFSarray[i].num_vertices;j++)
                    {
                          free(IFSarray[i].pointz);
                          free(IFSarray[i].indices);
                    }
              }

              free(IFSarray);
        }
        else
              return FALSE;
        isinitialized = FALSE;
        */
        return TRUE;
}

/*
This function loads the two java arrays with values from the C global
array.  It returns the number of vertices transfered.
*/

long CVMOObject_loadVMOObjectinC(struct HCVMOObject
*CVMOObjectthis,struct Hfloatarrayobject *floatdest,struct
Hintarrayobject *intdest)
{
        long i,j;
        long *Int_transferto;
        float *Flt_transferto;

        Int_transferto = (long *) intdest;
        Flt_transferto = (float *)floatdest;
```

```
	for(i = j = 2;i<(IFSarray[current_object].num_vertices + 2);i++){
		Int_transferto[i] = IFSarray[current_object].indices[i-2];
		Flt_transferto[j++] = IFSarray[current_object].pointz[i-
2].x;
		Flt_transferto[j++] = IFSarray[current_object].pointz[i-
2].y;
		Flt_transferto[j++] = IFSarray[current_object].pointz[i-
2].z;
	}
	current_object = ++current_object%num_objects;
	return i;
}
/*
	This function returns the number of vertices contained in the
object
that will be returned in a call to CVMOObject_loadVMOObjectinC.  This
makes it possible to dynamically allocate objects in java.
*/
long CVMOObject_getNuminNextVMOObjectinC(struct HCVMOObject
*CVMOObjectthis)
{
	if(isinitialized == FALSE)
		return -1;
	return IFSarray[current_object].num_vertices;
}
/*
	This function returns the number of the object that will be
returned
in a call to CVMOObject_loadVMOObjectinC.
*/
long CVMOObject_getNumofNextVMOObjectinC(struct HCVMOObject
*CVMOObjectthis)
{
	if(isinitialized == FALSE)
		return -1;
	return current_object;
}

#include <native.h>
/* Header for class CAvObject */
#ifndef _Included_CAvObject
#define _Included_CAvObject
#pragma pack(4)
typedef struct ClassCAvObject {
    char PAD;     /* ANSI C requires structures to have a least one
member */
} ClassCAvObject;
HandleTo(CAvObject);
#pragma pack()
#ifdef __cplusplus
extern "C" {
#endif
extern long CAvObject_getNuminNextAvObjectinC(struct HCAvObject
*,long);
extern long CAvObject_getNumofNextAvObjectinC(struct HCAvObject *);
extern long CAvObject_getStatusofObjectinC(struct HCAvObject *);
extern long CAvObject_setStatusofObjectinC(struct HCAvObject *,long);
```


Meta Exhibit 1011
Meta v. Mullen - Page 0085

```
extern long CAvObject_initAvObjectinC(struct HCAvObject *);
extern long CAvObject_destroyAvObjectinC(struct HCAvObject *);
extern float CAvObject_Cgetnextpointxfromc(struct HCAvObject *,long);
extern float CAvObject_Cgetnextpointyfromc(struct HCAvObject *,long);
extern float CAvObject_Cgetnextpointzfromc(struct HCAvObject *,long);
extern long CAvObject_Cgetnextindexfromc(struct HCAvObject *,long);
#ifdef __cplusplus
}
#endif
#endif


#include <StubPreamble.h>

/* Stubs for class CAvObject */
/* SYMBOL: "CAvObject/getNuminNextAvObjectinC(I)I",
Java_CAvObject_getNuminNextAvObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_getNuminNextAvObjectinC_stub(stack_item *_P_,struct
execenv *_EE_) {
      extern long CAvObject_getNuminNextAvObjectinC(void *,long);
      _P_[0].i =
CAvObject_getNuminNextAvObjectinC(_P_[0].p,((_P_[1].i)));
      return _P_ + 1;
}
/* SYMBOL: "CAvObject/getNumofNextAvObjectinC()I",
Java_CAvObject_getNumofNextAvObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_getNumofNextAvObjectinC_stub(stack_item *_P_,struct
execenv *_EE_) {
      extern long CAvObject_getNumofNextAvObjectinC(void *);
      _P_[0].i = CAvObject_getNumofNextAvObjectinC(_P_[0].p);
      return _P_ + 1;
}
/* SYMBOL: "CAvObject/getStatusofObjectinC()I",
Java_CAvObject_getStatusofObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_getStatusofObjectinC_stub(stack_item *_P_,struct
execenv *_EE_) {
      extern long CAvObject_getStatusofObjectinC(void *);
      _P_[0].i = CAvObject_getStatusofObjectinC(_P_[0].p);
      return _P_ + 1;
}
/* SYMBOL: "CAvObject/setStatusofObjectinC(I)I",
Java_CAvObject_setStatusofObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_setStatusofObjectinC_stub(stack_item *_P_,struct
execenv *_EE_) {
      extern long CAvObject_setStatusofObjectinC(void *,long);
      _P_[0].i = CAvObject_setStatusofObjectinC(_P_[0].p,((_P_[1].i)));
      return _P_ + 1;
}
/* SYMBOL: "CAvObject/initAvObjectinC()I",
Java_CAvObject_initAvObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_initAvObjectinC_stub(stack_item *_P_,struct execenv
*_EE_) {
```

```
        extern long CAvObject_initAvObjectinC(void *);
        _P_[0].i = CAvObject_initAvObjectinC(_P_[0].p);
        return _P_ + 1;
}
/* SYMBOL: "CAvObject/destroyAvObjectinC()I",
Java_CAvObject_destroyAvObjectinC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_destroyAvObjectinC_stub(stack_item *_P_,struct execenv
*_EE_) {
        extern long CAvObject_destroyAvObjectinC(void *);
        _P_[0].i = CAvObject_destroyAvObjectinC(_P_[0].p);
        return _P_ + 1;
}
/* SYMBOL: "CAvObject/CgetnextpointxfromC(I)F",
Java_CAvObject_CgetnextpointxfromC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_CgetnextpointxfromC_stub(stack_item *_P_,struct execenv
*_EE_) {
        extern float CAvObject_CgetnextpointxfromC(void *,long);
        _P_[0].f = CAvObject_CgetnextpointxfromC(_P_[0].p,((_P_[1].i)));
        return _P_ + 1;
}
/* SYMBOL: "CAvObject/CgetnextpointyfromC(I)F",
Java_CAvObject_CgetnextpointyfromC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_CgetnextpointyfromC_stub(stack_item *_P_,struct execenv
*_EE_) {
        extern float CAvObject_CgetnextpointyfromC(void *,long);
        _P_[0].f = CAvObject_CgetnextpointyfromC(_P_[0].p,((_P_[1].i)));
        return _P_ + 1;
}
/* SYMBOL: "CAvObject/CgetnextpointzfromC(I)F",
Java_CAvObject_CgetnextpointzfromC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_CgetnextpointzfromC_stub(stack_item *_P_,struct execenv
*_EE_) {
        extern float CAvObject_CgetnextpointzfromC(void *,long);
        _P_[0].f = CAvObject_CgetnextpointzfromC(_P_[0].p,((_P_[1].i)));
        return _P_ + 1;
}
/* SYMBOL: "CAvObject/CgetnextindexfromC(I)I",
Java_CAvObject_CgetnextindexfromC_stub */
__declspec(dllexport) stack_item
*Java_CAvObject_CgetnextindexfromC_stub(stack_item *_P_,struct execenv
*_EE_) {
        extern long CAvObject_CgetnextindexfromC(void *,long);
        _P_[0].i = CAvObject_CgetnextindexfromC(_P_[0].p,((_P_[1].i)));
        return _P_ + 1;
}
```


Meta Exhibit 1011
Meta v. Mullen - Page 0087

3 Java Code to Generate Virtual Masking Objects

```
import java.lang.*;
import java.applet.*;
import java.awt.*;
import java.util.*;
import vrml external.field.*;
import vrml external.Node;
import vrml.external.Browser;
import vrml.external.exception.*;
import netscape.javascript.JSObject;

public class AvatarClass extends Object
{
    Browser          ParentBrowser = null;
    Node[]           VRML_string;
    String           description;
    boolean          ReadyFlag = false;
    boolean          DebugToggle = true;
    EventInMFNode    addChildren;
    EventInMFNode    removeChildren;
    Node             root_node;
    String           aNodeName;
    CAvObject        ThePipe;//Communication link to data in "C"
    int              num_objects;

    public AvatarClass ( Browser aBrowser, String Node2Use,boolean DebugSwitch)
    {
        ParentBrowser = aBrowser;
        DebugToggle = DebugSwitch;
        aNodeName = Node2Use;
        if(DebugToggle == true)
        {
            System.out.println("Initializing the CAvObject..");
        }
        ThePipe = new CAvObject();
        num_objects = ThePipe.InitCAvObject();//returns the number of IFS's waiting to load.
        if(DebugToggle == true)
        {
            System.out.println("Initialized the CAvObject: " + num_objects);
        }
        if(ParentBrowser == null)
            ReadyFlag = false.
        else
        {
            ReadyFlag = setupnode();
        }
    }//End AvatarClass constructor

    //    Get the node from VRML, return true if found, otherwise false.

    public boolean setupnode ()
    {
        int NUM_RETRIES = 100;
```


Meta Exhibit 1011
Meta v. Mullen - Page 0088

```
        if(DebugToggle == true)
            System.out.println("Attempting to get the <IFS> node:" + aNodeName);

      /* If there is a node, this loop is normally executed once when opened
        from within Netscape and twice when double clicked or run from a shortcut. */

for(int i = 0; i < NUM_RETRIES; i++)
        {
            try { root_node = ParentBrowser.getNode(aNodeName);}
                catch(InvalidNodeException e){}
            if(root_node != null)  //Quit looping if node is found
            {
                break;
            }
            else  // otherwise, go to sleep for awhile and try again
            {
                if(DebugToggle == true)
                {
                    System.out.println("InvalidNodeException->Will try again.");
                }
                try   {    Thread.sleep(50);    }
                catch(InterruptedException e) {}
            }
        }
        if(root_node == null)  //Return false if a failure :-(
        {
            return false.
        }
          if(DebugToggle == true)   //print the information in Java console
        {
            System.out.println("- Got the ROOT node: " + root_node);
            String root_node_type = root_node.getType();
            System.out.println("  The node is of type '" + root_node_type + "'.");
            if (!root_node_type.equals("Group"))
            {
                System.out.println("WARNING!  initScene: ROOT node (type '"+
                                                        root_node_type+"')"+
                                                        " is not a 'Group'!");
            }
        }
        //Get the handles needed to add and abort children
        //Given the node, this succeeds the first time.
        try
        {
            addChildren = (EventInMFNode) root_node.getEventIn("addChildren");
            removeChildren = (EventInMFNode) root_node.getEventIn("removeChildren");
        }
    catch (InvalidEventInException e)
        {
            System.out.println("Could not access eventIn on root_node: " + e);
            return false;
        }
        return true;
    }//End setupnode()
    //Add the node to the current scene
    public boolean addChild()
```


Meta Exhibit 1011
Meta v. Mullen - Page 0089

```
{

        if (ReadyFlag == false)//Either no browser or node->sorry
                return false;
        if(DebugToggle == true)
        {
                System.out.println("Node to Create VRML:\n" + description);
        }
        //Attempt to convert the description String into a VRML node
        try {
                VRML_string = ParentBrowser.createVrmlFromString(description);
        }
        catch (InvalidVrmlException e)
        {
                System.out.println("Bad node description: InvalidVrmlException: " + e);
                return false;
        }
        //Add to the scene
  addChildren.setValue(VRML_string);
        return true;
  }//End addchild()

  //Delete the node from the current scene
public boolean removeChild()
  {
        if (ReadyFlag == false)//Either no browser or node->sorry
                return false;
        if(DebugToggle == true)
        {
                System.out.println("Node to delete VRML:\n" + description);
        }

        removeChildren.setValue(VRML_string);
        return true;
 }//End removeChild()

  //Load the Avatar node in memory into class
   public boolean load_descriptor()
   {

        int                     i, j, num_points;
        String string1;
        //logic here to determine status of avatars
        if((ReadyFlag == false)||(num_objects <= 0))
                return false;

        //Begin building the VRML compliant string.
                string1 =       "#VRML V2.0 utf8\n" +
                                "Transform {"+
                                " \n children Shape {\n" +
                                "    appearance Appearance {\n" +
                                "       material Material {\n" +
                                "                ambientIntensity  0\n"+
                "               diffuseColor      0 0 0\n"+
                "               shininess         0 \n"+
                                "       }\n" +
```

```
                    "    }\n" +
                    "geometry IndexedFaceSet { coord Coordinate { point[ ";
            description = "";
        for(i = 0;i < num_objects;i++)//for each IFS
        {
            num_points = ThePipe.getNuminNextAvObject(i);//Get the number of points in the current
IFS
            description += string1;
            if(DebugToggle == true)
            {
                System.out.println("Number of vertices: " + num_points);
            }
            for(j = 0;j<(num_points*3);j+=3)//Points listed one after the other, getnextpointxfromC
                                        //steps a pointer to the next float
            {
                description +=      ThePipe.getnextpointxfromC(i) + "   " +//Add each point
                                ThePipe.getnextpointyfromC(i) + "   " +
                                ThePipe.getnextpointzfromC(i) + ",\n";
            }
            description +=      "] } coordIndex [ ";//close the point description
            for(j = 0;j<num_points;j++)//get and add the indices
            {
                //description +=     ThePipe.getnextindexfromC(i) + ", ";
                description +=  j + ", ";
            }
            description +=      "0, -1 ] },"+//close the index description and the IFS
                                "  } },\n";

        }//End for

        return true;
    }//end load_descriptor()
}//End AvatarClass
```

4 Java Code to Get Laser Data Stream

```
public class IPRCtrls extends Applet implements EventOutObserver
{
  Browser        browser = null;
  AvatarClass                Addins;
        public int CURRENTLY_WRITING =      1;
        public int READY_TO_READ =          2;
        public int CURRENTLY_READING =      3;
        public int SUCCESSFULLY_READ =      4;
  public void init ()
  {
        Addins = new AvatarClass(browser, "ROOT",true);
              MyThread MT = new MyThread(this);
              MT.start();
    Echo("Finished initialization");
  }

  public void destroy()
  {
```

```
            if(Addins.ThePipe.getStatusofCAvObject() == SUCCESSFULLY_READ)
                    Addins.ThePipe.setStatusofCAvObject(READY_TO_READ);
            Addins.removeChild();
  }

class MyThread extends Thread{
   IPRCtrls the_applet;
   static long i = 0;
   static long SLEEP_DELAY = 500;

   MyThread(IPRCtrls the_applet) {
         this.the_applet = the_applet;
   }

   public void run() {

                  while (true) {
                  try {sleep(SLEEP_DELAY);}
                          catch (InterruptedException e) {}
                          if(the_applet.Addins.ThePipe.getStatusofCAvObject() ==
the_applet.READY_TO_READ )
                          {
                                  System.out.println("Setting to CURRENTLY_READING");

         the_applet.Addins.ThePipe.setStatusofCAvObject(the_applet.CURRENTLY_READING);
                                  the_applet.Addins.load_descriptor();
                                  the_applet.Addins.addChild();
                                  System.out.println("Setting to SUCCESSFULLY_READ");

         the_applet.Addins.ThePipe.setStatusofCAvObject(the_applet.SUCCESSFULLY_READ);
                                  System.out.println("Done");
                          }
                  }
   }
}

public class CAvObject
{

  public CVMOObject()
  {
    System.loadLibrary ("CAvObject");
  }


  public int InitAvObject()
  {
       return initAvObjectinC();

  }
  public int loadVMOObject (floatarrayobject floatdest, intarrayobject intdest)
  {
       return loadVMOObjectinC(floatdest, intdest);

  }
```

```
public int getNuminNextAvObject ()
{
      return getNuminNextAvObjectinC();

}
public int getNumofNextAvObject ()
{
      return getNumofNextAvObjectinC();

}
   public int destroyVMOObject ()
{
      return getNumofNextAvObjectinC();
  }
private native int loadAvObjectinC (floatarrayobject points, intarrayobject indices);
private native int getNuminNextAvObjectinC();
private native int getNumofNextAvObjectinC();
private native int initAvObjectinC();
private native int destroyAvObjectinC();
}
```


Meta Exhibit 1011
Meta v. Mullen - Page 0093

Appendix B - Example of Implementing Virtual Simulation

The code segments presented in this Appendix, written in the Hypertext Markup Language (HTML), JavaScript, Java, Virtual Reality Modeling Language (VRML), the VRML External Interface Specification, and VRMLScript will enable a programmer experienced in the above languages to reconstruct the Virtual Simulation and display rendering elements of the software supporting the reference apparatus. The section on "Zoom Gunsight Optics" also includes code to control a video camera observing the real-world scene.

This Appendix contains the following sections Page No.

1 Scenario Start Code 2
1.1 Relevant HTML Code 2
1.2 Relevant JavaScript Code 2
1.3 Relevant Java Code 2
1.4 Relevant VRML Code 3
2 Scenario Stop Code 5
2.1 Relevant HTML Code 5
2.2 Relevant JavaScript Code 5
2.3 Relevant VRML Code 6
3 Fire Main Gun 7
3.1 Relevant HTML Code 7
3.2 Relevant JavaScript Code 7
3.3 Relevant Java Code 7
3.4 Relevant VRML Code 7
4 Zoom Gunsight Optics 8
4.1 Relevant HTML Code 8
4.2 Relevant JavaScript Code 8
4.3 Relevant Java Code 9
4.4 Relevant VRML Code 11
5 Reticle Simulation 13
5.1 Relevant Java Code 13
5.2 Relevant VRML Code, Simulating a Gun Sight Reticle 13
6 Virtual T-80 Battle Tank Prototype File Listing 21
7 Implementation Notes 50

Meta Exhibit 1011
Meta v. Mullen - Page 0094

1 Scenario Start Code

The following code is used to start a scenario in the reference apparatus software.

1.1 Relevant HTML Code

Within a Forms construct, "Start" button starts scenario execution:

```
<TD>
    <center>
      <input type="Button"
          value="  Start   "
          onClick="setStart()">
      </Input>
      <br>
      <input type="Button"
          value=" Reset "
          onClick="setReset()">
      </Input>
    </center>
</TD>
```

1.2 Relevant JavaScript Code

Refers to Java applet "Interface"

```
function setStart() {
self.parent.frames[1].document.Interface.setStartMovement();
                    }
```

1.3 Relevant Java Code

```
// Note: assumes scenario has up to Number_of_Tanks virtual tanks
 public void setStartMovement()
{
    if (allowStart)
        {
        Echo("Start Movement Selected");
        for (int i=0; i<Number_of_Tanks; i++)
            if (Tank[i].isActive())
                Tank[i].setMovementStarted();
        allowStart = false;
        }
}
```

```
eResetInternals = (EventInSFBool) TankIs.getEventIn("ResetInternals");
eStartInternals  = (EventInSFBool) TankIs.getEventIn("StartInternals");
eTurretSweep   = (EventInSFBool) TankIs.getEventIn("TurretAutoSweep");
eMoveNow      = (EventInSFBool) BFCalc.getEventIn("moveNow");


public void setMovementStarted()
{
        eStartInternals.setValue(true);
        eTurretSweep.setValue(true);
        eMoveNow.setValue(true);
}
```

1.4 Relevant VRML Code

```
# Note: Defines reference to T-80 Main Battle Tank Prorotyppe

EXTERNPROTO t80Prototype [
        exposedField  SFVec3f    TheTranslation
        exposedField  SFRotation  TheRotation
        exposedField  SFVec3f    TheViewpoint
        exposedField  SFColor    TheBodyColor
        exposedField  SFFloat    TheBodyShininess
        exposedField  SFFloat    TheBodyAmbience
        exposedField  SFColor    TheSpecularColor
        exposedField  SFColor    TheEmissiveColor
        exposedField  SFFloat    TheTransparency
        exposedField  SFBool     RespondToTouch
        eventIn      SFBool     ResetInternals
        eventIn      SFBool     StartInternals
        eventIn      SFBool     StartTheHunt
        eventIn      SFBool     YouGotHit
        eventIn      SFFloat    SetGunElevation      # arg = elev, radians
        eventIn      SFBool     FireNow
        eventIn      SFBool     TurretAutoSweep
        eventIn      SFBool     StopGrass
        eventOut      SFVec3f    TouchedMeAt
        eventOut      SFBool     StopMyMotion
        eventOut      SFVec3f    PositionReport
                                ]  "t80proto.wrl"

DEF FieldOperations Script  {
        eventIn  SFBool     moveNow
```


Meta Exhibit 1011
Meta v. Mullen - Page 0096

```
eventIn  SFVec3f    NewViewAdjustment
eventIn  SFTime     speed
eventIn  SFFloat    setTankAmbientIntensity
eventIn  SFFloat    setTankShininess
eventIn  SFFloat    setTankTransparency
eventIn  SFColor    setTankDiffuseColor
eventIn  SFColor    setTankSpecularColor
eventIn  SFColor    setTankEmissiveColor
eventOut SFColor  UpdateDiffuseColor
eventOut SFColor  UpdateEmissiveColor
eventOut SFColor  UpdateSpecularColor
eventOut SFFloat  UpdateAmbientIntensity
eventOut SFFloat  UpdateShininess
eventOut SFFloat  UpdateTransparency

field   SFColor     black   0    0    0
field   SFColor     brown   0.8  0.7  0.4
field   SFColor     white   0.8  0.8  0.8

eventOut SFTime     startMovement
eventOut SFBool     enableMovement

field   SFNode    MyView   USE OP

directOutput   TRUE

url "vrmlscript:
function moveNow(value,time) {
                        if (value == TRUE)
                        {
                        enableMovement = true;
                        startMovement = time;
                        }
                }
```


Meta Exhibit 1011
Meta v. Mullen - Page 0097

2 Scenario Stop Code

The following code sequence stops scenario execution

2.1 Relevant HTML Code

Same as in 1.1. STOP Activated by Reset button.

2.2 Relevant JavaScript Code

```
function setReset() {
self.document.forms[0].ATCheck.checked = false;
self.parent.frames[1].document.Interface.setReset();
}
```

Relevant Java Code

```
Tank[0] = new TankClass(browser,
                "TankOne",
                "T1Route",
                "T1Rot",
                "T1Timer",
                getParameter("Tank1Track"),
                OkToEcho.booleanValue());
            public void setReset()
            {
                Echo("Reset Selected");
                setPauseMovement();

                for (int i=0; i< Number_of_Tanks; i++)
                    Tank[i].resetPosition();
                sHome.setValue(true);
                allowStart = true;
            }

            public void setPauseMovement()
            {
                Echo("Pause Movement Selected");
                for (int i=0; i<3; i++)
                    if (Tank[i].isActive()) Tank[i].setMovementPaused();
            }
```


Meta Exhibit 1011
Meta v. Mullen - Page 0098

```
Node TankIs = MyBrowser.getNode(aNodeName);
ePosition = (EventInSFVec3f) TankIs.getEventIn("set_TheTranslation");
eRotation = (EventInSFRotation) TankIs.getEventIn("set_TheRotation");
eTouchLocation = (EventOutSFVec3f) TankIs.getEventOut("TouchedMeAt");
eStopMotion = (EventOutSFBool) TankIs.getEventOut("StopMyMotion");
eMyPosition = (EventOutSFVec3f) TankIs.getEventOut("PositionReport");
eStartTracking = (EventInSFBool) TankIs.getEventIn("StartTheHunt");
eResetInternals = (EventInSFBool) TankIs.getEventIn("ResetInternals");

Node RtTimer = MyBrowser.getNode(aTimerName);
eMoveActive = (EventInSFBool) RtTimer.getEventIn("set_enabled");
eMoveActive.setValue(false);
eMoveCTime = (EventInSFTime) RtTimer.getEventIn("set_cycleInterval");
eMoveSTime = (EventInSFTime) RtTimer.getEventIn("set_startTime");
eMoveETime = (EventInSFTime) RtTimer.getEventIn("set_stopTime");

public void resetPosition()
        {
                if (tankActive)
                {
                        ePosition.setValue(MyPosition);
                        eRotation.setValue(MyRotation);
                }
                eResetInternals.setValue(true);
                setMovementPaused();
                setMovementTimer(speed_factor);
        }
public void setMovementPaused()
        {
                eMoveActive.setValue(false);
        }
public void setMovementTimer(float  aMovementTime)
        {
                // 36 kph or 10 mps IF quotient  = 10
                MyMovementTime = PathLength / aMovementTime;
                eMoveCTime.setValue(MyMovementTime);
                eMoveSTime.setValue(0);
                eMoveETime.setValue(0);
        }
```

2.3 Relevant VRML Code

Identical to code in Section 1.3

Meta Exhibit 1011
Meta v. Mullen - Page 0099

3 Fire Main Gun

The code implements the engagement of a virtual tank, currently tracked, from the main gun, as activated by the observer. The observer is at the simulated gunner's position of a simulated Main Battle Tank.

3.1 Relevant HTML Code

In Frame construct:

```
<input type="Button"
        value="    FIRE    "
        onClick="shootNow()">
 </Input>
```

3.2 Relevant JavaScript Code

```
function shootNow() {
        self.parent frames[1].document.Interface.setFire();
                }
```

3.3 Relevant Java Code

```
Node RetBB = browser.getNode("MOVEBB");
IFireNow = (EventInSFBool) RetBB.getEventIn("IFire");

        public void setFire()
        {
                Echo("Fire Selected");
                IFireNow.setValue(true);
        }
```

3.4 Relevant VRML Code

```
DEF MOVEBB Script  {
                eventIn SFBool          IFire
                eventOut SFVec3f        targetPosition
                eventOut SFTime         FireBigGun
                eventOut SFBool         hitTankOne
                eventOut SFBool         hitTankTwo
                eventOut SFBool         hitTankThree
                eventOut SFBool         huntTankOne
```

```
eventOut SFBool        huntTankTwo
eventOut SFBool        huntTankThree

  function IFire(value, time) {
 if (tracking == true)
 {
   FireBigGun = time;
   huntTankOne = true;
   huntTankTwo = true;
   huntTankThree = true;
   if (targetindex == 1) hitTankOne = true;
   if (targetindex == 2) hitTankTwo = true;
   if (targetindex == 3) hitTankThree = true;
 }
}
```

4 Zoom Gunsight Optics

The following code we sets the magnification (zoom levels) for the simulated tank optics in the VRML simulation through which the observer observers the combined real world plus virtual scene. It also sets the zoom level of the camera observing the real world.

4.1 Relevant HTML Code

In Forms construct:

```
<TD>
      <U>Zoom Control</U><br>
      <SELECT NAME="ZControl"
          SIZE="0"
          onChange="setZoom()"
          onBlur="ZControl.blur()">
          <OPTION VALUE="0">1.5x
          <OPTION VALUE="1" SELECTED>3x
          <OPTION VALUE="2">9.5x
          <OPTION VALUE="3">Laserx
      </SELECT>
    </TD>
```

4.2 Relevant JavaScript Code

```
function setZoom() {
// index = 0 - 1x, 3x, 2= 9.5x,  3 = LaserX
self.parent.zoomState = document.forms[0].ZControl.selectedIndex;
if (document.forms[0].ZControl.selectedIndex == 0)
  self.parent.frames[1].document.Interface.setZoom1x();
else if (document.forms[0].ZControl.selectedIndex == 2)
```


Meta Exhibit 1011
Meta v. Mullen - Page 00101

```
        self.parent.frames[1].document.Interface.setZoom9x();
    else if(document.forms[0].ZControl.selectedIndex == 1)
        self.parent.frames[1].document.Interface.setZoom3x();
                else
        self.parent.frames[1].document.Interface.setZoomLx();
    }
```

4.3 Relevant Java Code

```
    switch (VRMLStartAt)
    {
    case 0:  Node OP = browser.getNode("OP");
            FOV = (EventInSFFloat) OP.getEventIn("set_fieldOfView");
            VPRot = (EventInSFRotation) OP.getEventIn("set_orientation");
            VPPos = (EventInSFVec3f) OP.getEventIn("set_position");
            VRMLStartAt++;

    }

public void setZoom1x()
{
  Echo("1.5x Zoom Selected");
  CurrentZoom = 0;
  FOV.setValue(CurrentFOV[CurrentZoom]);
  Rfov.setValue(CurrentFOV[CurrentZoom]);
        GotoPreSetting(CurrentZoom);
        setPhotoHShift (0); //ksg
  setPhotoVShift (0);
}
public void setZoomLx()
{
  Echo("Laser Zoom Selected"),
  CurrentZoom = 3;
  FOV.setValue(CurrentFOV[CurrentZoom]);
  Rfov.setValue(CurrentFOV[CurrentZoom]);
        GotoPreSetting(CurrentZoom);
        setPhotoHShift (0); //ksg
  setPhotoVShift (0);
}
public void setZoom3x()
{
  Echo("3x Zoom Selected");
  CurrentZoom = 1;
  FOV.setValue(CurrentFOV[CurrentZoom]);
  Rfov.setValue(CurrentFOV[CurrentZoom]);
```

```
    GotoPreSetting(CurrentZoom);
    setPhotoHShift (0); //ksg
    setPhotoVShift (0);
  }

  public void setZoom9x()
  {
    Echo("9.5x Zoom Selected");
    CurrentZoom = 2;
    FOV.setValue(CurrentFOV[CurrentZoom]);
    Rfov.setValue(CurrentFOV[CurrentZoom]);
    GotoPreSetting(CurrentZoom);
    setPhotoHShift (0); //ksg
    setPhotoVShift (0);
  }
  public void GotoPreSetting(int zoomnum)
        {
                CurrentZoom = zoomnum;
                if(View.Cam_Command( ID_SET_SETZOOM ,
PreSettings[CurrentZoom][2])!= 0)
                        Echo("Zoomed");
                else
                        Echo("Didn't zoom Down");
                if(View.Cam_Command(ID_SET_SETPAN,
PreSettings[CurrentZoom][0])!= 0)
                        Echo("Panned");
                else
                        Echo("Didn't Pan");
                if(View.Cam_Command(ID_SET_SETTILT,
PreSettings[CurrentZoom][1])!= 0)
                        Echo("Tilted");
                else
                        Echo("Didn't Tilt");
                FOV setValue(CurrentFOV[CurrentZoom]);
                Rfov setValue(CurrentFOV[CurrentZoom]);

                setPhotoHShift (0); //ksg
                setPhotoVShift (0);
        }
          public void setPhotoHShift (int relativeShift) //ksg
  {
    Echo("Setting Photo Horizontal Shift : " + relativeShift + " " +
photo_H_Shift[CurrentZoom]);
    photo_H_Shift[CurrentZoom] += relativeShift;
          if(relativeShift < 0)
          {
```

```
            if(View.Cam_Command(ID_PAN_LEFT, -1 * relativeShift)!= 0)
                    Echo("Moved Left");
            else
                    Echo("Didn't move Left");
        }
        else if(relativeShift != 0)
        {
            if(View.Cam_Command(ID_PAN_RIGHT  ,relativeShift)!= 0)
                    Echo("Moved Right");
            else
                    Echo("Didn't move Right");
        }
    }

    public void setPhotoVShift (int relativeShift) //ksg
    {
      Echo("Setting Photo Vertical Shift : " + relativeShift + " " +
photo_V_Shift[CurrentZoom]);
      photo_V_Shift[CurrentZoom] += relativeShift;
      if(relativeShift < 0)
        {
            if(View.Cam_Command(ID_PAN_DOWN  ,-1*relativeShift)!= 0)
                    Echo("Moved Down");
            else
                    Echo("Didn't move Down");
        }
        else  if(relativeShift != 0)
        {
            if(View.Cam_Command(ID_PAN_UP ,relativeShift)!= 0)
                    Echo("Moved Up");
            else
                    Echo("Didn't move Up");
        }
    }
```

4.4 Relevant VRML Code

```
DEF OP Viewpoint {
        fieldOfView   0.0526  # 9.5x
        position    0   1.6   10
        description  "9.5x View"
}
Viewpoint {
        fieldOfView   0.16667  # 3x
        position    0   1.6   10
```


Meta Exhibit 1011
Meta v. Mullen - Page 00104

```
            description "3x View"
        }
        Viewpoint {
            fieldOfView  0.3226  # 1.55x
            position    0   1.6   10
            description "1.55x View"
        }
        Viewpoint {
            fieldOfView  0.119  # Laserx
            position    0   1.6   10
            description "Laser View"
        }


eventIn SFFloat        setFOV
eventOut SFFloat       zoomValue_changed
field  SFNode  view    USE OP


 function setFOV(value, time) {
   // moves billboard children into field of view
   ctf =  value*1 25;  // position & scale multiplier
   tsc.x = ctf;  // setting up scale
   tsc.y = ctf;
   tsc.z = 1;
   view.set_fieldOfView = value;
   zoomValue_changed = value;
   nn  = board.children.length; // how many elements to move
   for (i = 1; i < nn; ++i)
   {  // loop for each element except reticle
     zz = places[i].z;
     tdv = places[i].multiply(value/stdview);
     tdv.z = zz;
     nd = board children[i]; // must do it for each child
     nd.set_skala=tsc; // setting children's scale
     nd.set_positio=tdv; // setting position
   }
 }
```


Meta Exhibit 1011
Meta v. Mullen - Page 00105

5 Reticle Simulation

The following code defines and activates the simulated gun sight reticle of the observer, through which the observer observes the combined scene. The reticle defined in VRML is part of a simulated Main Battle Tank, parked at a stationary position.

5.1 Relevant Java Code

```
switch (VRMLStartAt)
    {
        case 2:  Node Ret = browser.getNode("RETICLE");
        Rfov = (EventInSFFloat) Ret.getEventIn("set_rfov");
        sHome = (EventInSFBool) Ret.getEventIn("setHome");
          }
```

5.2 Relevant VRML Code, Simulating a Gun Sight Reticle

```
PROTO M1A1RET [
  exposedField    SFColor    rcolor     1  1  1
  exposedField    SFFloat    rint       2.7
  exposedField    SFVec3f    positio    0  0  5
  exposedField    SFRotation rrot       0 -1  0  0
  exposedField    SFVec3f    skala      1  1  1
  exposedField    SFFloat    rfov       0.06
  exposedField    SFVec3f    vposition  0   1   10
  field        SFBool     ReportRange  FALSE
  eventIn        SFBool     StartInternals
  eventIn        SFBool     ResetInternals
  eventIn        SFBool     startAutotrack
  eventIn        SFBool     setHome
  eventIn        SFVec3f    trackPosition
  eventIn        SFVec3f    alignReticle
  eventOut       SFRotation getFieldRotation
  eventOut       SFVec3f    getVPPosition
  eventOut       SFRotation getVPRotation
  eventOut       SFRotation getReticleShift

]
{
  PROTO Thick_Circle
  [
```

```
  exposedField SFVec3f cscal    0.043633231 0.043633231 1 #  1 deg
  exposedField SFColor ccolor   1   1   1
  exposedField SFFloat cint     2.71
]
{
  Transform
  {
    translation 0   0   0
    rotation    0   -1  0   0
    scale IS cscal
    children
    [
      Transform
      {
        translation 0 1.5 0
        rotation    0 0   1   1.50796
        children
        [
        DEF rod_segment Shape
        {
          appearance Appearance {
            material Material {
              ambientIntensity IS cint
              transparency 0
              emissiveColor IS ccolor
            }
          }
          geometry Cylinder
          {
            height 1.35
            radius 0.45
          }
        }
        ]
      },
      Transform { translation 0 -1.5 0
              rotation    0  0   0   1.50796
              children    USE       rod_segment }
      Transform { translation  1.5  0   0
              children    USE       rod_segment }
      Transform { translation -1.5 0   0
              children    USE       rod_segment }
      Transform { translation 1.06 1.08 0
              rotation    0   0   1   0.7854
              children    USE       rod_segment }
      Transform { translation -1.06 1.08 0
```

```
            rotation    0   0   1  -0.7854
            children    USE      rod_segment }
      Transform { translation 1.06 -1.08 0
            rotation    0   0   1  -0.7854
            children    USE      rod_segment }
      Transform { translation -1.06 -1.08 0
            rotation    0   0   1   0.7854
            children    USE      rod_segment }
    ]
  } # end thick circle transform
} # end PROTO Thick_Circle

PROTO HMARK [
  exposedField SFVec3f   mtrans  0  0  0
  exposedField SFRotation mrot   0  0  1  0
  exposedField SFVec3f   mcent   0  0  0
  exposedField SFColor   mcolor  1  1  1
  exposedField SFFloat   mtransp 0.49
  exposedField SFFloat   mint    2.71
]
{
  Transform {   translation IS mtrans
    rotation IS mrot
    center  IS  mcent
    children
    Shape {
      appearance Appearance {
        material Material {
          ambientIntensity 2.71
          transparency IS mtransp
          diffuseColor  IS mcolor
          emissiveColor IS mcolor
        }
      } # end Material + App Node
      geometry Box { size 0.01  0.00085 0.00085 }

    }
  }
} # end PROTO HMARK

DEF RET Transform {
  translation IS   positio
  rotation   IS   rrot
  scale      IS   skala
  children [
  DEF TEMPUS TimeSensor{ }
```


Meta Exhibit 1011
Meta v. Mullen - Page 00108

```
DEF TA_TIMER TimeSensor { cycleInterval 1.5}
DEF INNERWORKS Transform {

  center IS vposition
  children [
    #DEF C2 Thick_Circle {cscal 0.0025  0.0025  1},  # 1 mr reticle circle
    DEF C2 Thick_Circle {cscal 0.00165  0.00165  1},  # 1 mr reticle circle
    DEF H1 HMARK {mtrans 0.02 0  0 mtransp 0},
    DEF H2 HMARK {mtrans -0.02 0  0 mtransp 0},
    DEF H3 HMARK {mtrans 0.0125 0.01 0 mtransp 0},
    DEF H4 HMARK {mtrans -0.0125 0.01 0 mtransp 0},
    DEF H5 HMARK {mtrans 0.0125 -0.01 0 mtransp 0},
    DEF H6 HMARK {mtrans -0.0125 -0.01 0 mtransp 0},
    DEF H7 HMARK {mtrans 0.02 0 0 mrot 0  0 1 1.5708 mcent -0.02 0 0
mtransp 0}
     DEF H8 HMARK {mtrans -0.02 0 0 mrot 0  0 1 1.5708 mcent 0.02 0 0
mtransp 0}
DEF DMARK Transform { translation 0 -0.0325 0
      scale    1   1   1
      children [
        Shape {  # distance mark
          appearance Appearance {
            material Material {
              ambientIntensity 2 7
              transparency 0 #0.49
              # diffuseColor IS rcolor
              emissiveColor IS rcolor
            }
          } # end Material + App Node
          geometry Box { size 0.0125 0.0014 0.0014}
        }
       Transform {
       translation 0.01  0  0
       children [
        Shape {  # distance mark
         appearance Appearance {
           material
             DEF TRACKLIGHT Material {
             ambientIntensity 2.7
             #transparency 0.49
             diffuseColor 1 0 0
             emissiveColor 1   0   0
           }
         } # end Material + App Node
         geometry Sphere { radius 0.002}
         }
```


Meta Exhibit 1011
Meta v. Mullen - Page 00109

```
    ]
    }
        ] },
] },   # end INNERWORKS

DEF RETSCRIPT Script {

field  SFNode  ret   USE   RET
field  SFNode  dmark  USE   DMARK
field  SFNode  c2   USE   C2
field  SFNode innergimbal USE INNERWORKS
field  SFNode tracklight  USE TRACKLIGHT
field  SFNode tempus      USE TEMPUS
field  SFNode ta_timer    USE TA_TIMER
#Muzzle Velocity:  5500 ft/second  (Sabot), 3735 ft/second (HEAT)
#Sabot = 1676 m/s  HEAT = 1138 m/s
field  SFFloat sabotspeed   1676
field  SFFloat heatspeed    1138
field  SFVec3f reticlepos  0  0  5
field  SFVec3f dscale95      1  1  1
field  SFVec3f dtrans95     0  -0.0325  0
field  SFFloat retfov  0.06

field  SFBool  reportRange IS ReportRange
eventIn SFVec3f setRPos  IS   set_positio
eventIn SFFloat setZoom  IS   set_rfov
eventIn SFColor setColor IS   set_rcolor
eventIn SFVec3f provideVPxyz   IS set_vposition
eventIn SFVec3f followTarget   IS trackPosition
eventIn SFBool  startAutoTrack IS startAutotrack
eventIn SFBool  resetInternals IS ResetInternals
eventIn SFBool  startInternals IS StartInternals
eventIn SFBool          goHome        IS setHome
eventOut SFRotation      fieldRotation  IS getFieldRotation
#  reticle  constants and variables:

   field  SFVec3f  reticleposition  0     1    5
   field  SFColor  trackingcolor    0     1    0
   field  SFColor  notrackingcolor  1     0    0
   field  SFColor  transitioncolor  1     1    0
   field  SFVec3f  newdirection     0     0    -100
   field  SFVec3f  olddirection     0     0    -100
   field  SFVec3f  vpxyz            0     0    10
   field  SFVec3f  trackoffset      0    -2    0
   field  SFVec3f  targetxyz        0     0    -100
   field  SFVec3f  nullpozicio      0     0    0
```

```
        }
function startInternals(value, time) {
        if (value == true) initialize();
        }
function setZoom(value, time) {
        // needs to set scale factor for reticle
        // this is not linear with zoom fov = value!
        //m = value/0.03966;
        m = 1.0;
        if ((value >= 0.11) && (value < 0.3)) m = 9.5/3;
        if (value >= 0.3) m = 9.5;
        dmark.set_scale = dscale95.multiply(m);
        dmark.set_translation = dtrans95.multiply(m);
        retfov = value;
        }
function setRPos(value, time) {
        reticlepos = value;
        reticlepos.z = 5;
        ret.set_translation = reticlepos;
        }
function setColor(value, time) {
        c2.set_ccolor = value;
        }
function rotateReticle(xx, yy, rr) {
        elevation = yy/rr;
        azimuth   = -xx/rr;
        totalis   = Math.sqrt(elevation*elevation + azimuth*azimuth);
        yaxis =   azimuth/totalis;
        xaxis =   elevation/totalis;
        reticleRotation = new SFRotation(xaxis,yaxis,0,totalis);
        innergimbal.set_rotation = reticleRotation;
        }
   function followTarget(value, time) {
        if (tracking == true ) {
          targetxyz = value.add(trackoffset);
          newdirection = targetxyz.subtract(vpxyz);
          range = newdirection.length();
          if (!nowtracking) {
              oldtargdirection = value;
              tempus.set_startTime = time;
              ctmr =  ta_timer.fraction_changed;
              rotateReticle(newdirection.x*ctmr, newdirection.y*ctmr, range);
              if (ctmr > 0.9) {
                  nowtracking = true;
                  tracklight.set_diffuseColor = trackingcolor;
                  tracklight.set_emissiveColor = trackingcolor;
```

```
                }
            }
        else
        {
        xtime = time - tempus.startTime;
        newtime = xtime;
        dx = newdirection.x - oldtargdirection.x;
        vx = 0;
        dt = newtime-oldtime;
        if (dt > 0) vx = dx/dt;
        raiseCross(range,vx);
        rotateReticle(newdirection.x, newdirection.y, range);
        oldtime = newtime;
        oldtargdirection = newdirection;
        if (reportRange)  {
            s = 'Range: ' + Math.round(range/10)*10 + ' m';
            Browser setDescription(s);
            }
         } // end nowtracking
      }   // end tracking == true
    }
function startAutoTrack (value, time) {
      tracking = value;
      if ( value == true ) {
        oldtime = time;
        nowtracking = false;
        tracklight.set_diffuseColor = transitioncolor;
        tracklight.set_emissiveColor = transitioncolor;
        ta_timer set_startTime = time;
        }
      else {
        tracklight.set_diffuseColor = notrackingcolor;
        tracklight.set_emissiveColor = notrackingcolor;
        nowtracking = false,
      }
    }

    function provideVPxyz(value, time) {
      vpxyz = value.
    }

  "
  }
 ]
}
```

```
} # end M1A1RET PROTO
```

6 Virtual T-80 Battle Tank Prototype File Listing

The appearance (shape, colors) and internal behavior (turret animation, main gun animation, main gun fire, explosion in response to hit, start, stop, position reporting, animation of interface with real-world grass surface, indication of surface point coordinates touched to support software development) for any T-80 Main Battle Tank in the virtual simulation of the reference apparatus is defined in this prototype file).

```
#VRML V2.0 utf8

PROTO t80Prototype [
      exposedField  SFVec3f        TheTranslation  0 0 0
      exposedField  SFRotation     TheRotation    0 -1 0 0
      exposedField  SFVec3f        TheViewpoint   0  2  10
      exposedField  SFColor        TheBodyColor   0.447059 0.478431 0.196078
      exposedField  SFFloat        TheBodyShininess  0.2
      exposedField  SFFloat        TheBodyAmbience  0.2
      exposedField  SFColor        TheSpecularColor  0 0 0
      exposedField  SFColor        TheEmissiveColor  0 0 0
      exposedField  SFFloat        TheTransparency  0
      exposedField  SFBool         RespondToTouch   FALSE
      eventIn     SFBool     ResetInternals
      eventIn     SFBool     StartInternals
      eventIn     SFBool     StartTheHunt
      eventIn     SFBool     YouGotHit
      eventIn     SFFloat    SetGunElevation # arg = elev, radians
      eventIn     SFBool     FireNow
      eventIn     SFBool     TurretAutoSweep
      eventIn     SFBool     StopGrass
      eventOut    SFVec3f    TouchedMeAt
      eventOut    SFBool     StopMyMotion
      eventOut    SFVec3f    PositionReport
   ]
# --- tank tread is at -2 46 meters below z = 0 -------------
{
  PROTO flashProto [eventIn SFTime now
        exposedField SFVec3f        positio 0 0 0
        exposedField SFRotation     rotatio 0 0 -1 0
        exposedField SFVec3f        skala  1 1 1
        exposedField SFTime         duratio  0.8
  ]
  {
    Transform {
      translation IS positio
```

```
rotation IS rotatio
scale IS skala
children [
  NavigationInfo {
    headlight      FALSE
  }
  DEF FIRECONE Transform{
    rotation 0  0  -1   1.5708
    scale 0  0  0
    children
    Shape{
      appearance Appearance {
        texture PixelTexture {
          image 1 6 3 0xFF0000 0xFF8000
                  0xFFC000 0xFFFF00
                  0xFFFFC0 0xFFFFFF
        }
      }
      geometry Cone{}
    }
  }
  DEF FLASH PointLight {
    ambientIntensity  1
    intensity                 1
    location        0 0 0
    on             FALSE
    radius          100
  }
  DEF SHAPER PositionInterpolator {
    key        [0     0.25  0.5  0 75  0 85  1]
    keyValue    [0     0     0,
              0 5   1     0.5,
              1     2     1,
              1     4     1,
              1     8     1,
              0     0     0]
  }
  DEF FIRETIMER TimeSensor {
    cycleInterval IS duratio # seconds
    enabled TRUE
    loop FALSE
  }
  DEF FIREGUN Script {
    eventIn SFTime  shoot IS now
    eventIn SFVec3f fireNow
    eventOut SFTime trigger
```


Meta Exhibit 1011
Meta v. Mullen - Page 00114

```
        field SFNode fireball USE FIRECONE
        field SFNode light USE FLASH
        directOutput TRUE
        url "vrmlscript:
          function shoot(value,time){
            trigger = value;
          }

          function fireNow(value,time) {
            if (value.y > 0) {
              light.set_on = TRUE,
              u = light.location;
              u.x = -1*value.y;
              light.set_location = u;
            }
            else light.set_on = FALSE;
            fireball set_scale = value;
            xx = fireball.translation;
            xx.x = -1*value.y;
            fireball set_translation = xx;
          }
        "
      } # end of SCRIPT
    ]
  }
  ROUTE FIREGUN.trigger                TO    FIRETIMER.startTime
  ROUTE FIRETIMER.fraction_changed     TO    SHAPER.set_fraction
  ROUTE SHAPER.value_changed           TO    FIREGUN.fireNow
}
DEF T80_WORLD Transform {
  translation        IS      TheTranslation
  rotation           IS      TheRotation
  center             0       -2.46   0
  children [
    DEF HIT_FLASH PointLight {
        ambientIntensity  1
        intensity         1
        color             1 0.75 0.75
        location          0 0   0
        on               FALSE
        radius            100
    }
    DEF TARGETSEEK TimeSensor {   # find next target
      cycleInterval 10  # seconds
      loop TRUE
      enabled FALSE        }
```

```
DEF TurretTimer1 TimeSensor {
  cycleInterval 6   # for 180 degrees
}
DEF GunTubeTimer TimeSensor { # gun elevation timer
  cycleInterval 3
  loop FALSE
}
DEF TurretTimer TimeSensor { # turret auto rotation
  cycleInterval 30  # seconds
  enabled FALSE
  loop TRUE
}
DEF ReloadTimer TimeSensor { # turret auto rotation
  cycleInterval 6  # seconds
  enabled FALSE
  loop TRUE
}
DEF GrassTimer TimeSensor { # grass skirt driver
  cycleInterval 5  # seconds
  enabled TRUE
  loop TRUE
}
DEF GrassShaker PositionInterpolator {
  key [ 0    1 ]
  keyValue [0 -0.5 0,  0   0 5 0]
}
DEF HitFlash TimeSensor
{
  cycleInterval 2
  enabled      TRUE
  loop         FALSE
}
Sound {
  minFront 0.1
  minBack 0.1
  maxFront 1000
  maxBack 1000
  source  DEF BOOM AudioClip{url "boom.wav"}
}
DEF QuickRotation OrientationInterpolator {
  key [ 0    1 ]
  keyValue [0 -1  0  0,  0  -1   0   1]
}
DEF TurretHeading OrientationInterpolator {
  key  [    0       0.04     0.08333  0.125    0.16667
          0 20833  0.25     0.29167  0.33333  0.375
```


Meta Exhibit 1011
Meta v. Mullen - Page 00116

```
        0.41667 0.45833 0.5     0.54167 0.58333
        0.625   0.66667 0.70833 0.75    0.79167
        0.83333 0.875   0.91667 0.95833 1
      ]
  keyValue [ 0 -1 0 0
        0 -1 0 0.1309 #0.261799
        0 -1 0 0.2618 #0.523599
        0 -1 0 0.3927 #0.785398
        0 -1 0 0.5236 #1.047197
        0 -1 0 0.6545 #1.308996
        0 -1 0 0.7854 #1.570796

        0 -1 0 0.6545 #1.308996
        0 -1 0 0.5236 #1.047197
        0 -1 0 0 3927 # 0.785398
        0 -1 0 0 2618 #0 523599
        0 -1 0 0.1309 #0 261799
        0 -1 0 0

        0 -1 0 -0.1309 # -0.261799
        0 -1 0 -0 2618 #-0.523599
        0 -1 0 -0.3927 #-0 785398
        0 -1 0 -0.5236 #-1.047197
        0 -1 0 -0.6545 #-1.308996
        0 -1 0 -0.7854 #-1.570796

        0 -1 0 -0.6545 #-1 308996
        0 -1 0 -0.5236 #-1.047197
        0 -1 0 -0 3927 #-0 785398
        0 -1 0 -0.2618 #-0.523599
        0 -1 0 -0 1309 #-0 261799
        0 -1 0 -0
      ]
}
DEF TRANSITION OrientationInterpolator {
  key [0   1]
  keyValue [0 0 -1 0.04, 0 0 -1 0.1]
}
#Main Tank Definition Section
DEF T80 Transform {
  # Starting the World Transform at 0"
  children [
    DEF TankTouch TouchSensor { enabled IS RespondToTouch }
    DEF TankView Viewpoint {
      description "Open Hatch"
      position    1 10625        0 503501        -0 77373235
```

```
  orientation 0      -1      0      -1.5708
}
DEF Turret Transform {
  translation 0 -0.15 0
  rotation   0 -1 0 0
  center     1 0 0
  children [
    Transform {
      translation  1.02125 -0.927501 -0.0937323
      rotation 1  0  0   1.5708
      children [
        Transform {
          translation 0.3  0  -0.2
          scale  1.44  1.34  0.45
          children
          [
            Shape { # the cupola is a flattened sphere here..
              appearance Appearance {
                material Material {
                  diffuseColor        IS TheBodyColor
                  ambientIntensity    IS TheBodyAmbience
                  shininess           IS TheBodyShininess
                  specularColor       IS TheSpecularColor
                  emissiveColor       IS TheEmissiveColor
                  transparency        IS TheTransparency
                }
              }
              geometry Sphere { }
            },
            Transform { children [ # the 6 ERA blocks
              Transform
              {  # one block of ERA
                translation 0 19 0.95 -0.1 # rear left top
                children
                [
                  DEF era Shape
                  {
                    appearance Appearance {
                      material Material {
                        diffuseColor        IS TheBodyColor
                        ambientIntensity    IS TheBodyAmbience
                        shininess           IS TheBodyShininess
                        specularColor       IS TheSpecularColor
                        emissiveColor       IS TheEmissiveColor
                        transparency        IS TheTransparency
                      }
```

```
            }
            geometry Box {size 0.2 0.05 0.7 }
          }
        ]
      },
      Transform { translation -0.05 0.97 -0.1 # mid left top
            children  USE era },
      Transform { translation -0.29 0.95 -0.1 # front left top
            children  USE era },
      Transform { translation 0.19 -0.95 -0.1 # rear right top
            children  USE era },
      Transform { translation -0.05 -0.98 -0.1 # mid right top
            children  USE era },
      Transform { translation -0.29 -0.95 -0.1 # front right top
            children  USE era }
    ] }
  ]
},
Transform {
  translation  0.5  -0.6  -0.5
  rotation -1  0  0   1.5708
  children
    Shape {  # hatch replaced by cylinder
      appearance Appearance {
        material Material {
          diffuseColor        IS TheBodyColor
          ambientIntensity    IS TheBodyAmbience
          shininess           IS TheBodyShininess
          specularColor       IS TheSpecularColor
          emissiveColor       IS TheEmissiveColor
          transparency        IS TheTransparency
        }
      }
      geometry Cylinder {radius 0.37 height 0.3 }
    }
},
Transform {
  translation  0 65  0.6  -0.5  # aft left  down
  rotation -1  0  0   1.5708
  children
    Shape { # hatch replaced by cylinder
      appearance Appearance {
        material Material {
          diffuseColor IS TheBodyColor
          ambientIntensity IS TheBodyAmbience
          shininess IS TheBodyShininess
```


Meta Exhibit 1011
Meta v. Mullen - Page 00119

```
                specularColor IS TheSpecularColor
                emissiveColor IS TheEmissiveColor
                transparency  IS TheTransparency
              }
            }
            geometry Cylinder {radius 0.37 height 0.3 }
          }
        },
      Transform {
        translation  2.1    0     -0.4
        rotation  0   0   1    0 #1.5708
        children [
          Shape {# Horizontal cyl. fuel tanks, vertical hook on back end
            appearance Appearance {
              material Material {
                diffuseColor IS TheBodyColor
                ambientIntensity IS TheBodyAmbience
                shininess IS TheBodyShininess
                specularColor IS TheSpecularColor
                emissiveColor IS TheEmissiveColor
                transparency  IS TheTransparency
              }
            }
            geometry Cylinder {radius 0.22 height  1.8}
          }
        ]
    },
    Transform {
        translation  1.55    0.9     -0 35
        rotation  0   0   1    0
        children [
      DEF STRUT  Shape {#bustle fuel tank strut
      appearance Appearance {
        material Material {
          diffuseColor        IS TheBodyColor
          ambientIntensity    IS TheBodyAmbience
          shininess           IS TheBodyShininess
          specularColor       IS TheSpecularColor
          emissiveColor       IS TheEmissiveColor
          transparency        IS TheTransparency
        }
      }
      geometry Box {size 1  0.02  0.1 }
    }
  ]
},
```


Meta Exhibit 1011
Meta v. Mullen - Page 00120

```
Transform {
      translation  1.55    -0.9      -0.35
        rotation  0   0   1    0
        children  USE STRUT
},
Transform {
      translation  1.55    0.9      -0.15
        rotation  0   1   0    0.25
        children  USE STRUT
},
Transform {
      translation  1.55    -0.9      -0.15
        rotation  0   1   0    0.25
        children  USE STRUT
},
        DEF HITFLASH flashProto {
          positio 0 0  0
          rotatio 0 -1 0  2
          skala   1 3 3
          duratio  2
        }
        DEF SMALL_FLASH flashProto {
          positio 0  0  0.5
          rotatio 0 -1 0  1.507
          skala   0 2 2 2
          duratio  1
        }
      ]
    }

    Transform {
      translation 1 02125 -0.927501 -0.0937323
      rotation    0.577317 0.577317 0.577417  4.18889
      children [
        Transform {
          children [
            Transform { # commander's MG
              translation  -0.55    -0.15    0.73
              children [
                Shape {
                  appearance Appearance {
                    material Material {
                      diffuseColor  0.1 0.12  0.05
                    }
                  }
                  geometry Box {size 0.1 0.5 0.1}
```

```
            }
          ]
        }
        Transform { # commander's MG
          translation -0.55   -0.9   0.73
          children [
            Shape {
              appearance Appearance {
                material Material {
                  diffuseColor  0.1 0.12  0.05
                }
              }
              geometry Box {size 0.04 1 0.04}
            }
          ]
        }
        Transform { # commander's MG
          translation -0.55   -0.15   0.63
          children [
            Shape {
              appearance Appearance {
                material Material {
                  diffuseColor  0.1 0.12  0.05
                }
              }
              geometry Box {size 0.1 0.2 0.5 }
            }
          ]
        }
      ]
    }
  ]
}
DEF BIGGUN Transform {
    rotation 0  0  -1  0.01
    center  -0.5  -0.6  0
    translation 0 3 0 0 05
    children  [
      DEF Guntube Transform {
        translation 0 0 0
        rotation   0 0 0 0
        children [
          Transform {
            translation 1.02125 -0.927501 -0.0937323
            rotation   0.577317 0.577317 0.577417  4.18889
            children [
```

```
    Transform {
      translation  0  -3.44  0.28  # left aft  up
      rotation -1  0  0  0
      children
        Shape {  # gun tube replaced by cylinder
          appearance Appearance {
            material Material {
              diffuseColor        IS TheBodyColor
              ambientIntensity    IS TheBodyAmbience
              shininess           IS TheBodyShininess
              specularColor       IS TheSpecularColor
              emissiveColor       IS TheEmissiveColor
              transparency        IS TheTransparency
            }
          }
          geometry Cylinder {radius 0.1 height 5 }
        }
    }
  ]
}
Transform {
  translation 1 02125 -0.927501 -0.0937323
  rotation    0.577317 0.577317 0.577417  4.18889
  children [
    Transform {
      translation  0  -4.21  0.28  # left aft  up
      rotation -1  0  0  0
      children
        Shape {  # gun supressor replaced by cylinder
          appearance Appearance {
            material Material {
              diffuseColor        IS TheBodyColor
              ambientIntensity    IS TheBodyAmbience
              shininess           IS TheBodyShininess
              specularColor       IS TheSpecularColor
              emissiveColor       IS TheEmissiveColor
              transparency        IS TheTransparency
            }
          }
          geometry Cylinder {radius 0.13 height 1 }
        }
    }
  ]
}
Transform {
  translation 1.02125 -0.927501 -0.0937323
```


Meta Exhibit 1011
Meta v. Mullen - Page 00123

```
            rotation   0.577317 0.577317 0.577417  4.18889
            children [
              Transform {
                translation  0  -1.34  0.28  # left aft  up
                rotation 1  0  0   3.14
                children
                  Shape {  # gun pintle replaced by cone
                    appearance Appearance {
                      material Material {
                        diffuseColor IS TheBodyColor
                        ambientIntensity IS TheBodyAmbience
                        shininess IS TheBodyShininess
                        specularColor IS TheSpecularColor
                        emissiveColor IS TheEmissiveColor
                        transparency  IS TheTransparency
                      }
                    }
                    # geometry Cylinder {radius 0.14 height 0.68 }
                    geometry Cone {bottomRadius 0.28 height 2}
                  }
              }
            ]
          }
        ]
      }

      DEF GUNFLASH flashProto {
        positio -4.7 -0.65 -0.1
      }
    ]
  }
]
}

# Starting the Children of the World at 5/8"
Transform {    #grass
  translation 1.02125 -0.927501 -0 0937323
  rotation   0 577317 0.577317 0.577417  4.18889
    children   [
      DEF GRASS Transform {
        translation 0  0  -0 2
          children [
          DEF TrackGrass Shape {        #right track grass skirt
            appearance Appearance {
              material Material {
                ambientIntensity 0
```

```
      diffuseColor 0   0   0
    } # end Material Node
  } # end Appearance Node
  geometry  IndexedFaceSet {
    coord  Coordinate {
      point [
        -2   -2.1  -1.47,
        -2   -2.0  -1.17,  #1
        -2   -1.9  -1.47,

        -2   -1.7  -1.47,  #3
        -2   -1.5  -1.27,
        -2   -1 35 -1.47,  #5

        -2   -1    -1.47,
        -2   -0.95 -1.17,  #7
        -2   -0 9  -1.47,

        -2   -0.6  -1.47,  #9
        -2   -0.5  -1.27,
        -2   -0.4  -1.47,  #11

        -2    0    -1.47,
        -2    0.03 -1.17,  #13
        -2    0.12 -1.47,

        -2    0 48 -1.47,  #15
        -2    0 59 -1.27,
        -2    0 70 -1 47,  #17

        -2    1.0  -1.47,
        -2    1 05 -1 17,  #19
        -2    1.19 -1.47,

        -2    1 40 -1.47,  #21
        -2    1.43 -1.2,
        -2    1.49 -1.47,  #23

        -2    1 55 -1.47,  #24
        -2    1.73 -1.27,
        -2    1.91 -1.47,  #26

        -2    2.2  -1.47,  #27
        -2    2.3  -1.17,
        -2    2 4  -1.47,  #29
      ]
```

```
    } # end Coordinate
    coordIndex [
      0, 1, 2, -1,  3, 4, 5, -1,
      6, 7, 8, -1, 9, 10, 11, -1,
      12, 13, 14, -1, 15, 16, 17,  -1,
      18, 19, 20, -1, 21, 22, 23, -1,
      24, 25, 26, -1, 27, 28, 29, -1,
    ]
  } # end IndexedFaceSet
} # end Shape

Transform {
  rotation 0 0 -1 3.14159
  children
    USE TrackGrass
}
Shape {     #front track grass skirt
  appearance Appearance {
    material Material {
      ambientIntensity 0
      diffuseColor 0   0   0
    } # end Material Node
  } # end Appearance Node
  geometry  IndexedFaceSet {
    coord  Coordinate {
    point [
      1.45  -2.5  -1.47,
      1.40  -2.5  -1.17,  #1
      1.35  -2.5  -1.47,

      1 15  -2 5  -1.47,  #3
      1.05  -2.5  -1.27,
      1.00  -2.5  -1.47,   #5

      -0 95  -2 5  -1 47,
      -0.99  -2 5  -1.17,   #7
      -1.05  -2.5  -1.47,

      -1.3   -2.5  -1.47,   #9
      -1.32  -2 5  -1.27,
      -1.35  -2.5  -1.47,   #11
    ]
  } # end Coordinate
  coordIndex [
    0, 1, 2, -1,  3, 4, 5, -1,
    6, 7, 8, -1, 9, 10, 11, -1,
```

```
            ]
          } # end IndexedFaceSet
        } # end Shape
      ]
    } #end GRASS
  ]
} # end grass transform

Transform { # Hull
  translation 1.02125 -0.927501 -0.0937323
  rotation   0.577317 0.577317 0.577417  4.18889
  children [
    Transform {
      translation  0    -1.02    0.92
      children [
        Shape { # Hull
          appearance Appearance {
            material Material {
              diffuseColor          IS TheBodyColor
              ambientIntensity      IS TheBodyAmbience
              shininess             IS TheBodyShininess
              specularColor         IS TheSpecularColor
              emissiveColor         IS TheEmissiveColor
              transparency          IS TheTransparency
            }
          }
          geometry IndexedFaceSet {
            coord Coordinate {
              point [
                1    -0.97  -1.1  , #0
                1    -1.22  -1.18 , #1
                0.92 -1.22  -1.18 , #2
                0.92 -1.9   -1.6  , #3
                0.92 -1.42  -1.96 , #4
                0.92  3.96  -1.96 , #5
                0.92  4.35  -1.81 , #6
                0.92  4.65  -1.18 , #7
                1     4.35  -1.18 , #8
                1     4.35  -1.05 , #9
                1     2.73  -1.08 , #10
               -1    -0.97  -1.1  , #11
               -1    -1.22  -1.18 , #12
               -0.92 -1.22  -1.18 , #13
               -0.92 -1.9   -1.6  , #14
               -0.92 -1.42  -1.96 , #15
               -0.92  3.96  -1.96 , #16
```

```
            -0.92  4.35  -1.81 , #17
            -0.92  4.65  -1.18 , #18
            -1     4.35  -1.18 , #19
            -1     4.35  -1.05 , #20
            -1     2.73  -1.08 , #21
          ]
        }
        coordIndex [
          0, 1, 8, 9, 10, -1, # left top
          1, 2, 7, 8, -1,     # left cut
          2, 3, 4, 5, 6, 7, -1 # left bottom
          21, 20, 19, 12, 11, -1, # right top
          19, 18, 13, 12, -1,     # right cut
          18, 17, 16, 15, 14, 13, -1, # right bottom
          0, 11, 12, 1, -1,    # front top slope
          2, 13, 14, 3, -1, # front top slope lower
          3, 14, 15, 4, -1, # front bottom slope
          4, 15, 16, 5, -1, # main bottom
          5, 16. 17, 6, -1, # aft bottom
          6, 17, 18, 7, -1, # lower tail
          8, 19, 20, 9, -1, # upper tail
          0, 10, 21, 11, -1, # top plate
          10, 9, 20, 21, -1, # engine cover
        ]
        normalPerVertex FALSE
        solid          FALSE
        creaseAngle    0.5
      }
    }
  ]
}
]
}

Transform {
  translation 1 02125 -0 927501 -0 0937323
  rotation   0.577317 0 577317 0.577417  4.18889
  children [
    DEF mud_guard Transform {
      children [
        Shape { # Right side mudguard
          appearance Appearance {
            material Material {
              diffuseColor          IS TheBodyColor
              ambientIntensity      IS TheBodyAmbience
              shininess             IS TheBodyShininess
```

```
    specularColor        IS TheSpecularColor
    emissiveColor        IS TheEmissiveColor
    transparency         IS TheTransparency
  }
}
geometry IndexedFaceSet {
  coord Coordinate {
    point [
      -1.6383 -2.0955 -0.25106,  # 0
      -1.6383  3.0162 -0.3095,   # 1
      -1.6383  3 0162 -0.2306,   # 2
      -1.6383 -2.0447 -0.2536,   # 3
      -1.6383  0      -0.7854,   # 4
      -1.6383 -2.9083 -0.6854,   # 5
      -1.6383 -2.1971 -0.24852,  # 6
      -1.6383 -2.9654 -0.6838,   # 7
      -1 6383 -2.921  -0.6076,   # 8
      -1.6383 -2.4765 -0.2663,   # 9
      -1.6383 -2.8512 -0.4314,   # 10
      -1.6383 -2.6619 -0.30948,  # 11
      -1.6383 -2.7686 -0.36155,  # 12
      -1.6383 -2.667  -0.25995,  # 13
      -1.6383 -2.4765 -0 24598,  # 14
      -1.6383 -2 794  -0.2917,   # 15
      -1 6383 -2.9083 -0.36155,  # 16
      -1 6383 -2.9908 -0.4568,   # 17
      -1 6383 -3 048  -0.6838,   # 18
      -0.9119 -2.9908 -0.4568,   # 19
      -0.9119 -3.048  -0.6838,   # 20
      -0.9119 -2.9083 -0.36155,  # 21
      -0 9119 -2.794  -0.2917,   # 22
      -0.9119 -2 667  -0.25995,  # 23
      -0 9119 -2.4765 -0.24598,  # 24
      -0.9119 -2.1971 -0.24852,  # 25
      -0.9119 -2.3292 -0.32472,  # 26
      -0.9119 -2.9159 -0.67778,  # 27
      -1.0033 -2.1971 -0.24852,  # 28
      -1.0033  3 3274 -0.2333,   # 29
      -1.6383  3.4925 -0.2533,   # 30
      -1 0033  3.4925 -0.2533,   # 31
      -1 6383  2 1717 -0 31456,  # 32
      -1 6383  2 1717 -0.35012,  # 33
      -1.6383  3.0162 -0.2552,   # 34
      -1.6383 -2.0955 -0.22566,  # 35
      -1.6383  1.6383 -0.3044,   # 36
      -1.6383  1.6383 -0.3298,   # 37
```

```
    -1.2954 -2.1273 -0.1266,   # 38
    -1.0033 -2 1273 -0.1266,   # 39
    -1.5875 -2.1273 -0.24852,  # 40
    -1.5875 -2.1273 -0.22312,  # 41
    -1.0033 -2.1273 -0.24852,  # 42
    -1.2954 -0.9677 -0.1266,   # 43
    -1.0033 -0.9677 -0.1266,   # 44
    -1.5875 -0.9677 -0.27265,  # 45
    -1.5875 -0.9677 -0.24725,  # 46
    -1.0033 -0.9677 -0.27265,  # 47
    -1.5875  0.0762 -0.2917,   # 48
    -1.0033  0 0762 -0.2536,   # 49
    -1.0033  0.0762 -0.2917,   # 50
    -1.5875  0.0762 -0.2663,   # 51
    -1.143   0.0762 -0.24598,  # 52
    -1.2878  0.0762 -0.152,    # 53
    -1.3716  0 0762 -0.1266,   # 54
    -1.3005  0.0762 -0.1266,   # 55
    -1.5875  3.0099 -0.21266,  # 56
    -1.0033  3.0099  0.0266,   # 57
    -1.1176  3.0099  0.0266,   # 58
    -1.5875  3.0099 -0.2306,   # 59
    -1.0033  3 0099 -0.2306,   # 60
    -1.6383  3 6576 -0.8203,   # 61
    -1.6383  3.5877 -0.6822,   # 62
    -1.6383  3 6195 -0.6949,   # 63
    -1.6383  3 5433 -0.6695,   # 64
    -0.9144  3 6576 -0.8203,   # 65
    -0.9144  3 4925 -0.2533,   # 66
    -1.6383  3 4925 -0.8854,   # 67
    -1.6383  3 4925 -0.6695    # 68
  ]
}
coordIndex [
  0, 1, 2, 3, -1,
  0, 4, 1, -1,
  0, 5, 4, -1,
  0, 6, 7, 5, -1,
  6, 8, 7, -1,
  6, 9, 10, 8, -1,
  9, 11, 12, 10, -1,
  13, 9, 6, 14, -1,
  12, 11, 9, 13, -1,
  15, 8, 10, 13, -1,
  16, 7, 15, -1,
  16, 17, 18, 7, -1,
```

```
            18, 17, 19, 20, -1,
            16, 21, 19, 17, -1,
            15, 22, 21, 16, -1,
            13, 23, 22, 15, -1,
            14, 24, 23, 13, -1,
            6, 25, 24. 14, -1,
            26, 24, 25, -1,
            26, 22, 23, 24, -1,
            26, 19, 21, 22, -1,
            26, 27, 20, 19, -1,
            0, 3, 28, 6, -1,
            3, 29, 28. -1,
            3, 30, 31. 29, -1,
            32, 33, 2, 34, -1,
            35, 0, 3, 36, -1,
            3, 37, 36, -1,
            48, 49, 50, -1,
            48, 51, 52, 49. -1,
            61, 30, 62, 63, -1,
            62, 30, 64, -1,
            30, 2, 1, 64, -1,
            30, 61, 65, 66, -1,
            67, 68, 1, 4. -1.
            50, 49, 48, -1,
            49, 52, 51, 48, -1,
          ]
          normalPerVertex FALSE
          solid          FALSE
          creaseAngle    0.5
        }
      }
    ]
  }
]
}

Transform {
  translation 1.02125 -0 927501 -0.0937323
  rotation    0.577317 0 577317 0.577417  4.18889
  children [
    DEF mud_guard Transform {
      children [
        Shape { # Left side mudguard
          appearance Appearance {
            material Material {
              diffuseColor            IS TheBodyColor
```

```
    ambientIntensity      IS TheBodyAmbience
    shininess             IS TheBodyShininess
    specularColor         IS TheSpecularColor
    emissiveColor         IS TheEmissiveColor
    transparency          IS TheTransparency
  }
}
geometry IndexedFaceSet {
  coord Coordinate {
    point [
      1.6383 -2.0955 -0.25106,  # 0
      1.6383  3.0162 -0.3095,   # 1
      1.6383  3.0162 -0.2306,   # 2
      1.6383 -2.0447 -0.2536,   # 3
      1.6383  0      -0.7854,   # 4
      1.6383 -2.9083 -0.6854,   # 5
      1.6383 -2.1971 -0.24852,  # 6
      1.6383 -2.9654 -0.6838,   # 7
      1.6383 -2.921  -0.6076,   # 8
      1.6383 -2.4765 -0.2663,   # 9
      1.6383 -2.8512 -0.4314,   # 10
      1.6383 -2.6619 -0.30948,  # 11
      1.6383 -2.7686 -0.36155,  # 12
      1.6383 -2.667  -0.25995,  # 13
      1.6383 -2.4765 -0.24598,  # 14
      1.6383 -2.794  -0.2917,   # 15
      1.6383 -2.9083 -0.36155,  # 16
      1.6383 -2.9908 -0.4568,   # 17
      1.6383 -3.048  -0.6838,   # 18
      0.9119 -2.9908 -0.4568,   # 19
      0.9119 -3.048  -0.6838,   # 20
      0.9119 -2.9083 -0.36155,  # 21
      0.9119 -2.794  -0.2917,   # 22
      0.9119 -2.667  -0.25995,  # 23
      0.9119 -2.4765 -0.24598,  # 24
      0.9119 -2.1971 -0.24852,  # 25
      0.9119 -2.3292 -0.32472,  # 26
      0.9119 -2.9159 -0.67778,  # 27
      1.0033 -2.1971 -0.24852,  # 28
      1.0033  3.3274 -0.2333,   # 29
      1.6383  3.4925 -0.2533,   # 30
      1.0033  3.4925 -0.2533,   # 31
      1.6383  2.1717 -0.31456,  # 32
      1.6383  2.1717 -0.35012,  # 33
      1.6383  3.0162 -0.2552,   # 34
      1.6383 -2.0955 -0.22566,  # 35
```


Meta Exhibit 1011
Meta v. Mullen - Page 00132

```
    1.6383  1.6383 -0.3044,    # 36
    1.6383  1.6383 -0.3298,    # 37
    1.2954 -2.1273 -0.1266,    # 38
    1.0033 -2.1273 -0.1266,    # 39
    1.5875 -2.1273 -0.24852,   # 40
    1.5875 -2.1273 -0.22312,   # 41
    1.0033 -2.1273 -0.24852,   # 42
    1.2954 -0.9677 -0.1266,    # 43
    1.0033 -0.9677 -0.1266,    # 44
    1.5875 -0.9677 -0.27265,   # 45
    1.5875 -0.9677 -0.24725,   # 46
    1.0033 -0.9677 -0.27265,   # 47
    1.5875  0.0762 -0.2917,    # 48
    1.0033  0.0762 -0.2536,    # 49
    1.0033  0.0762 -0.2917,    # 50
    1.5875  0.0762 -0 2663,    # 51
    1.143   0 0762 -0.24598,   # 52
    1.2878  0.0762 -0.152,     # 53
    1.3716  0.0762 -0.1266,    # 54
    1.3005  0.0762 -0.1266,    # 55
    1.5875  3.0099 -0.21266,   # 56
    1.0033  3.0099  0.0266,    # 57
    1.1176  3.0099  0.0266,    # 58
    1.5875  3.0099 -0.2306,    # 59
    1.0033  3.0099 -0 2306,    # 60
    1.6383  3.6576 -0 8203,    # 61
    1.6383  3.5877 -0.6822,    # 62
    1.6383  3.6195 -0.6949,    # 63
    1.6383  3.5433 -0.6695,    # 64
    0.9144  3 6576 -0 8203,    # 65
    0 9144  3 4925 -0 2533,    # 66
    1.6383  3.4925 -0 8854,    # 67
    1.6383  3.4925 -0.6695     # 68
  ]
}
coordIndex [
  0, 1, 2, 3, -1,
  0, 4, 1, -1,
  0, 5, 4, -1,
  0, 6, 7, 5, -1,
  6, 8, 7, -1,
  6, 9, 10, 8. -1,
  9, 11, 12, 10, -1,
  13, 9, 6. 14, -1,
  12, 11, 9, 13, -1,
  15. 8, 10, 13. -1.
```


Meta Exhibit 1011
Meta v. Mullen - Page 00133

```
                16, 7, 15, -1,
                16, 17, 18, 7, -1,
                18, 17, 19, 20, -1,
                16, 21, 19, 17, -1,
                15, 22, 21, 16, -1,
                13, 23, 22, 15, -1,
                14, 24, 23, 13, -1,
                6, 25, 24, 14, -1,
                26, 24, 25, -1,
                26, 22, 23, 24, -1,
                26, 19, 21, 22, -1,
                26, 27, 20, 19, -1,
                0, 3, 28, 6, -1,
                3, 29, 28, -1,
                3, 30, 31, 29, -1,
                32, 33, 2, 34, -1,
                35, 0, 3, 36, -1,
                3, 37, 36, -1,
                48, 49, 50, -1,
                48, 51, 52, 49, -1,
                61, 30, 62, 63, -1,
                62, 30, 64, -1,
                30, 2, 1, 64, -1,
                30, 61, 65, 66, -1,
                67, 68, 1, 4, -1,
                50, 49, 48, -1,
                49, 52, 51, 48, -1,
              ]
              normalPerVertex FALSE
              solid          FALSE
              creaseAngle    0.5
            }
          }
        ]
      }
    ]
  }
  Transform {
    translation 1.02125 -0.927501 -0.0937323
    rotation    0 577317 0 577317 0 577417  4.18889
    children [
      Transform {
        translation  0    3.52     0
        rotation  0   0   1    1.5708
        children [
          Shape {# external fuel tanks
```

```
                appearance Appearance {
                  material Material {
                    diffuseColor          IS TheBodyColor
                    ambientIntensity      IS TheBodyAmbience
                    shininess             IS TheBodyShininess
                    specularColor         IS TheSpecularColor
                    emissiveColor         IS TheEmissiveColor
                    transparency          IS TheTransparency
                  }
                }
                geometry Cylinder {radius 0.25 }
              }
            ]
          }
          Transform {
            translation  0    3 55     -0.5
            rotation  0   0   1    1.5708
            children [
              Shape {# Horizontal cyl. exhaust below fuel tanks, vertical hook on back
end
                appearance Appearance {
                  material Material {
                    diffuseColor        IS TheBodyColor
                    ambientIntensity    IS TheBodyAmbience
                    shininess           IS TheBodyShininess
                    specularColor       IS TheSpecularColor
                    emissiveColor       IS TheEmissiveColor
                    transparency        IS TheTransparency
                  }
                }
                geometry Cylinder {radius 0 17 height  1.8}
              }
            ]
          }
        ]
      }
      Transform {  # Left tread by IJ
          translation 1.02125 -0.927501 -0.0937323
          rotation    0.577317 0 577317 0 577417  4.18889
          children [
            DEF tread Transform {
              translation  0    -1.02    0 92
              children [
                Shape { # Hull
                  appearance Appearance {
                    material Material {
```

```
            diffuseColor 0.1  0.1  0.1

          }
        }
        geometry IndexedFaceSet {
          coord Coordinate {
            point [
              1.52  4.28  -1.38 ,  #0 left out top back
              1.52  4.28  -1.81 ,  #1 left out btm back
              0.97  4.28  -1.81 ,  #2 left in btm back
              0.97  4.28  -1.38 , #3 left in top back
              0.97  3.38  -2.45 ,  #4 left in grnd back
              1.52  3.38  -2.45 ,  #5  left out grnd back
              1.52  -0.88  -2.45 ,  #6  left out grnd fwd
              0.97  -0.88  -2.45 ,  #7   left in grnd fwd
              1.52  -1.83  -1.90 ,  #8   l. out low fwd
              0.97  -1.83  -1.90 ,  #9   l. in low fwd
              1.52  -1.83  -1.51 ,  #10   l. out hi fwd
              0.97  -1.83  -1.51 ,  #11   l. in hi fwd
              1.52 -1.7 -1.48 ,     #12   l fwd end out
              0.97 -1.7 -1.48 ,     #13   l fwd end in
            ]
          }
          coordIndex [
            0, 1, 2, 3, -1,  # left back tread
            2, 1, 5, 4, -1,   # left back slope
            4, 5, 6, 7, -1,  # left bottom
            7, 6, 8, 9, -1,  # left fwd bottom slope
            10, 11, 9, 8, -1, # left fwd face
            10, 12, 13, 11, -1, # left top
          ]
          normalPerVertex FALSE
          solid          FALSE
          creaseAngle     0.5
        }
      }
    ]
  }
]
}

Transform {  # Right tread by IJ
    translation 1.02125 -0.927501 -2.5937323
    rotation    0.577317 0.577317 0.577417  4.18889
    children USE tread
}
```


Meta Exhibit 1011
Meta v. Mullen - Page 00136

```
Transform {  #The Wheel Assembly
  translation 1.02125 -0.927501 -0.0937323
  rotation    0.577317 0.577317 0.577417  4.18889
  children [
    Transform  {
     translation 1.28 -1 79   -1.15
     rotation  0   0   1    1 5708
     children
      DEF MAINWHEEL Shape {
           appearance Appearance {
            material Material {
                 diffuseColor  0.1 0.12  0 08  }
                 }
           geometry Cylinder {radius 0 32 height 0.5 }
           }
     }
      Transform { # left wheel no. 2 from front
       translation   1.28    -1 06      -1.15
       rotation  0   0   1    1 5708
       children
         USE MAINWHEEL
     }
     Transform { # left wheel # 3
       translation   1.28    -0 18      -1 15
       rotation  0   0   1    1 5708
       children
         USE MAINWHEEL
     }
     Transform { # left wheel # 4
       translation   1.28    0 57      -1.15
       rotation  0   0   1    1 5708
       children
         USE MAINWHEEL
     }
     Transform { # left wheel # 5
       translation   1.28    1 44      -1.15
       rotation  0   0   1    1 5708
       children
         USE MAINWHEEL
     }
     Transform { # left wheel # 6
       translation   1.28    2 3      -1 15
       rotation  0   0   1    1 5708
       children
         USE MAINWHEEL
     }
```

```
Transform { # left (rear ) drive wheel cylinder
  translation  1.28   3 02    -0.84
  rotation  0   0   1    1 5708
  children
    DEF aSmallWheel Shape {
      appearance Appearance {
        material Material {
          diffuseColor  0 1 0.12  0.08
        }
      }
      geometry Cylinder {radius 0 23 height 0.5 }
    }
}
Transform { # left (front ) idler wheel cylinder
  translation  1.28   -2 63    -0.72
  rotation  0   0   1    1 5708
  children
    USE aSmallWheel
}
Transform { # RIGHTt wheel no. 1 from front
  translation  -1.28   -1.79    -1.15   #-1.88 FWD?
  rotation  0   0   1    1 5708
  children
    USE MAINWHEEL
}
Transform { # right wheel no  2 from front
  translation  -1.28   -1 06    -1 15
  rotation  0   0   1    1 5708
  children
    USE MAINWHEEL
}
Transform { # right wheel # 3
  translation  -1.28   -0 18    -1 15
  rotation  0   0   1    1 5708
  children
    USE MAINWHEEL
}
Transform { # right wheel # 4
  translation  -1.28   0 57    -1.15
  rotation  0   0   1    1 5708
  children
    USE MAINWHEEL
}
Transform { # right wheel # 5
  translation  -1.28   1 44    -1.15
```


Meta Exhibit 1011
Meta v. Mullen - Page 00138

```
        rotation  0   0   1    1.5708
        children
          USE MAINWHEEL
      }
      Transform { # right wheel # 6
        translation  -1.28    2 3     -1.15   #2.21?
        rotation  0   0   1    1.5708
        children
          USE MAINWHEEL
      }

      Transform { # right (rear ) drive wheel cylinder
        translation  -1.28    3 02     -0.84
        rotation  0   0   1    1.5708
        children
          USE aSmallWheel
      }
      Transform { # right (front ) idler wheel cylinder
        translation  -1.28    -2 63     -0.72
        rotation  0   0   1    1 5708
        children
          USE aSmallWheel
      }
    ]
  }
 ]  # Ending the Children of the World at 5/8"
}

DEF DATAOUT Script {
  eventIn  SFVec3f   touchedAt
  eventIn  SFFloat   gunElevation      IS SetGunElevation
  eventIn  SFBool    shootNow          IS FireNow
  eventIn  SFBool    hitResponse       IS YouGotHit
  eventIn  SFBool    turretSweep       IS TurretAutoSweep
  eventIn  SFVec3f   posChanged        IS set_TheTranslation
  eventIn  SFVec3f   vpChanged         IS set_TheViewpoint
  eventIn  SFBool    reset             IS ResetInternals
  eventIn  SFBool    startup           IS StartInternals
  eventIn  SFBool    startHunt         IS StartTheHunt
  eventIn  SFBool    stopGrass         IS StopGrass
  eventIn  SFTime    aimAtOP
  eventOut SFVec3f   touchPoint        IS TouchedMeAt
  eventOut SFBool    stopMotion        IS StopMyMotion
  eventOut SFVec3f   reportPos         IS PositionReport
  eventOut SFTime    pullTrigger
  eventOut MFString  Position
```


Meta Exhibit 1011
Meta v. Mullen - Page 00139

```
eventOut SFTime     soundStart
eventOut SFTime     smallStart
eventOut SFTime     explosionStart
eventOut SFBool     StopTheGrass
field   SFVec3f     opvp       0  2  10
field   SFRotation  trot       0 -1 0 0
field   SFRotation  aleph      0 0 0 0
field   SFVec3f     gcenter    -0.5  -0.6  0
field   SFRotation  gelev      0  0 -1  0.01
field   SFRotation  newelev    0  0  -1  0.1
field   SFRotation  gunbearing     0  -1 0  0
field   SFRotation  opbearing      0 -1 0 0
field   SFBool      turretIsSweeping  TRUE
field   SFBool      hunting  FALSE
field   SFBool      stopped FALSE
field   SFString    pos  ""
#====nodes in fields==========
field SFNode turm          USE Turret
field SFNode gun           USE BIGGUN
field SFNode guntimer      USE GunTubeTimer
field SFNode transition    USE TRANSITION
field SFNode turrettimer   USE TurretTimer
field SFNode grasstimer    USE GrassTimer
field SFNode t80tank       USE T80_WORLD
field SFNode flasher       USE HitFlash
directOutput TRUE
url "vrmlscript
  function roundDig(value,digits) {
    z = Math.pow(10,digits);
    zr = Math.round(z*value)/z;
    return zr;
  }
  function signum(value) {
    s = 0;
    if (value > 0) s = 1;
    if (value < 0) s = -1;
    return s;
  }

  function startHunt (value, time){ }
  function vpChanged (value, time){ }
  function reset(value, time) { }
  function startup(value, time) { }
  function posChanged(value, time) {
  }
```


Meta Exhibit 1011
Meta v. Mullen - Page 00140

```
    function shootNow(value,time) {
      if (value == TRUE)
        pullTrigger = time;
    }

    function turretSweep (value, time) {
      turrettimer.set_cycleInterval = 30 + (10 * (0.5 - Math.random()));
      turrettimer.set_startTime = time + (Math.random() * 15);
      turrettimer.set_enabled = value;
    }

    function touchedAt (value, time ) {
      Browser.setDescription ('x='+value.x+', y='+value.y+', z='+value.z);
      touchPoint = value;
    }

    function hitResponse (value, time) {
      if (Math.random() > 0 2)
      {
        flasher.set_startTime = time,
        stopMotion = true;
        grasstimer set_enabled = false.
        stopped = true;
        turrettimer set_enabled = false;
        if (Math.random() > 0.7)
          { explosionStart = time, }
        else
          { smallStart = time, }
      }
    }

    function gunElevation(value.timevalue){
      gelev.angle = gun.rotation.angle,
      newelev.angle = value.
      kV = new MFRotation ( gelev, newelev);
      transition set_keyValue = kV;
      guntimer set_startTime = timevalue,
    }

    function stopGrass (value, time) {
      StopTheGrass = !value;
    }
   "
  }
]
}
```

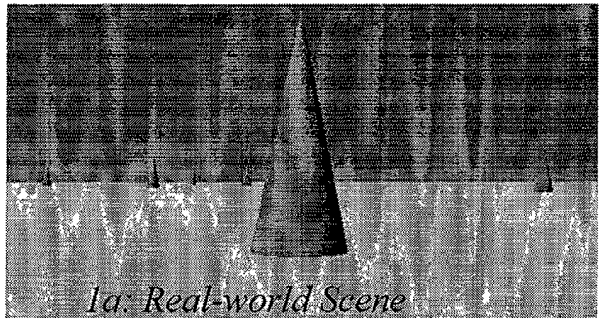
```
  ROUTE DATAOUT.pullTrigger TO GUNFLASH.now
  ROUTE DATAOUT.pullTrigger          TO BOOM.set_startTime
  ROUTE TankTouch.hitPoint_changed   TO DATAOUT.touchedAt
  ROUTE GunTubeTimer.fraction_changed TO TRANSITION.set_fraction
  ROUTE TRANSITION.value_changed     TO BIGGUN.set_rotation
  ROUTE DATAOUT smallStart           TO SMALL_FLASH.now
  ROUTE DATAOUT.explosionStart       TO HITFLASH.now
  ROUTE DATAOUT.explosionStart       TO BOOM.startTime
  ROUTE TurretTimer.fraction_changed TO TurretHeading.set_fraction
  ROUTE TurretHeading.value_changed  TO Turret.set_rotation
  ROUTE GrassTimer fraction_changed  TO GrassShaker.set_fraction
  ROUTE GrassShaker.value_changed    TO GRASS.set_translation
  ROUTE DATAOUT.StopTheGrass         TO GrassTimer.set_enabled
  ROUTE HitFlash isActive            TO HIT_FLASH.set_on
}
```

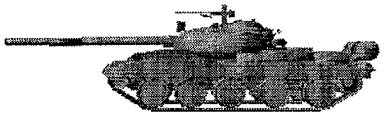
7 Implementation Notes

Note 1: Implementation of the T-80 Virtual Prototype and surrounding virtual simulation environment requires a VRML 2.0 compatible browser (interpretation environment).

Note 2: Certain VRML browsers implement only a Javascript-equivalent subset of the full VRMLScript syntax as defined in the VRML 2.0 standard. This model requires a fully VRML 2.0-compatible VRMLScript implementation.

Meta Exhibit 1011
Meta v. Mullen - Page 00142





1b: Virtual Image

1c: Masked Virtual Image

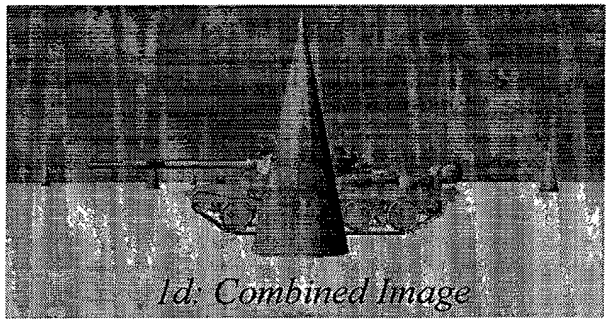


Figure 1

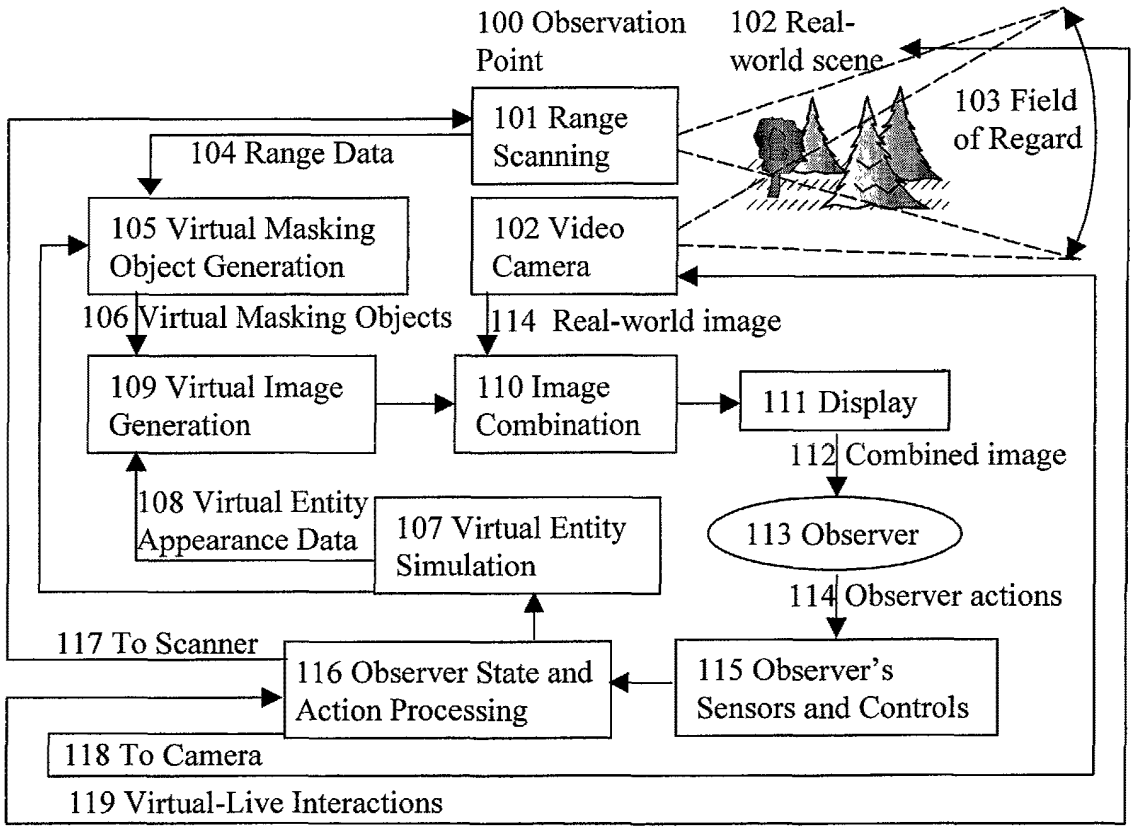

100 Observation Point
102 Real-world scene
103 Field of Regard
101 Range Scanning
104 Range Data
105 Virtual Masking Object Generation
102 Video Camera
106 Virtual Masking Objects
114 Real-world image
109 Virtual Image Generation
110 Image Combination
111 Display
112 Combined image
108 Virtual Entity Appearance Data
107 Virtual Entity Simulation
113 Observer
114 Observer actions
117 To Scanner
116 Observer State and Action Processing
115 Observer's Sensors and Controls
118 To Camera
119 Virtual-Live Interactions

Figure 2

Figure 3: Laser-generated Range Map

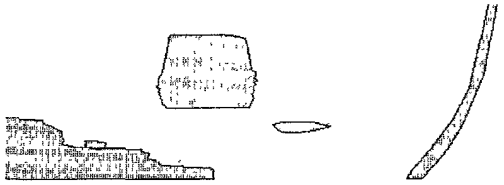


Figure 4: Virtual Masking Objects

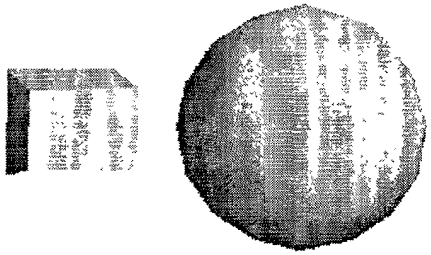


Figure 6: Effect of virtual masking object

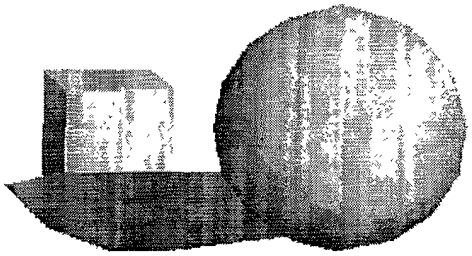


Figure 7: Virtual masking object is shown

```
#VRML V2.0 utf8
Transform {
 children Shape {
   appearance Appearance {
     material Material {
                                ambientIntensity  0
                                diffuseColor      0 0 0
                                shininess         0
     }
   }
geometry IndexedFaceSet { coord Coordinate { point[
        -0.0655473   1.14116    -45.9981,
        -0.235112    1.10839    -44.9971,
        -0.106866    0.980146   -44.9962,
         0.062694    0.952124   -43.9958,
         0.229877    0.952132   -43.9952,
         0.564238    0.993969   -43.9928,
         0.748038    1.02297    -44.9906,
         0.939429    1.05388    -45.9877,
         0.58991     1.1412     -45.9943,
] } coordIndex [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 0, -1 ] },  } }
```

Figure 5: Virtual masking object display constructs, VRML

$$A_n(t) = [A_n(T_1) - A_n(T_0)](t - T_1), \quad n = 1 \ldots N$$
$$E_n(t) = [E_n(T_1) - E_n(T_0)](t - T_1), \quad n = 1 \ldots N$$
$$r_n(t) = [r_n(T_1) - r_n(T_0)](t - T_1), \quad n = 1 \ldots N$$

Figure 8: Linear extrapolation of vertex positions between range frame acquisitions.

```
For each display row i {
    For each pixel j {
        Render_real_pixel = true;
        For each attribute k of the pixel in the virtual image {
            Transparent_background = ((background_attribute(k) == attribute(k));
            Render_real_pixel  = Render_real_pixel && Transparent_background;
        }
        If (Render_real_pixel ) Show the digital video pixel;
        Else Show the virtual display pixel;
    }
}
```

Figure 10: Pseudo-code for Digital Image Combination

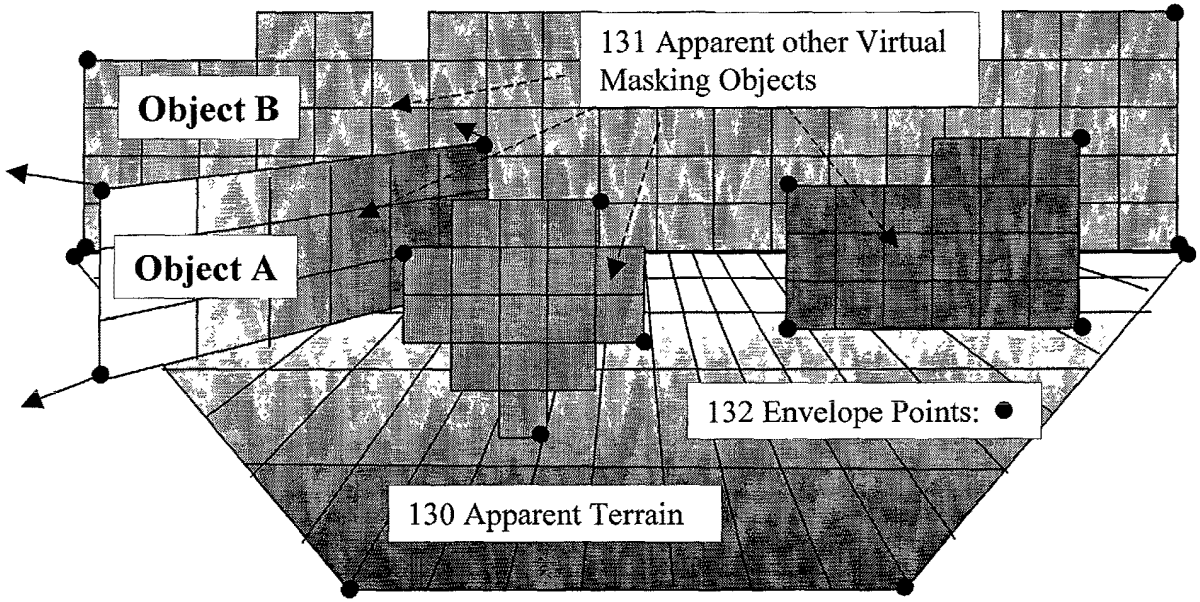


Figure 9: Virtual Masking Objects with Envelope Points Used in Extrapolation Logic

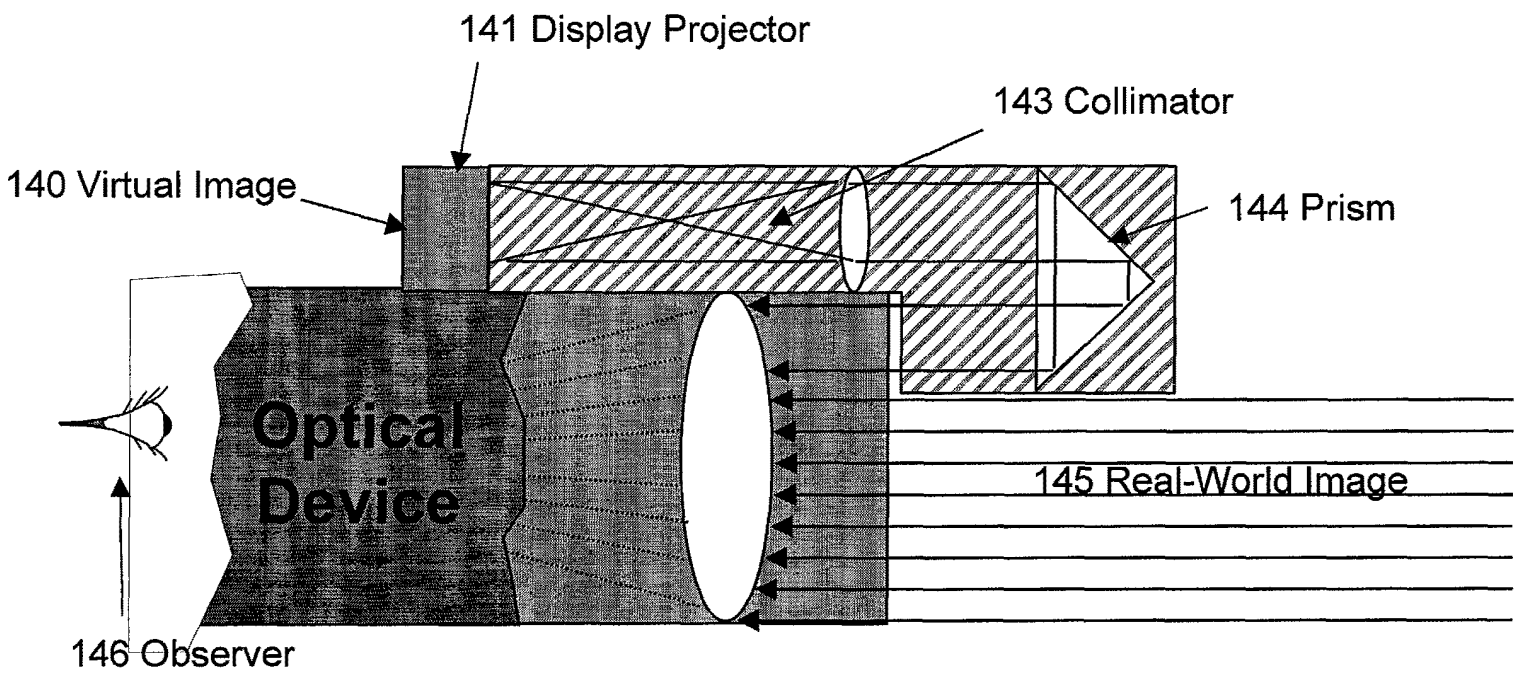


Figure 11: Virtual Image Injection

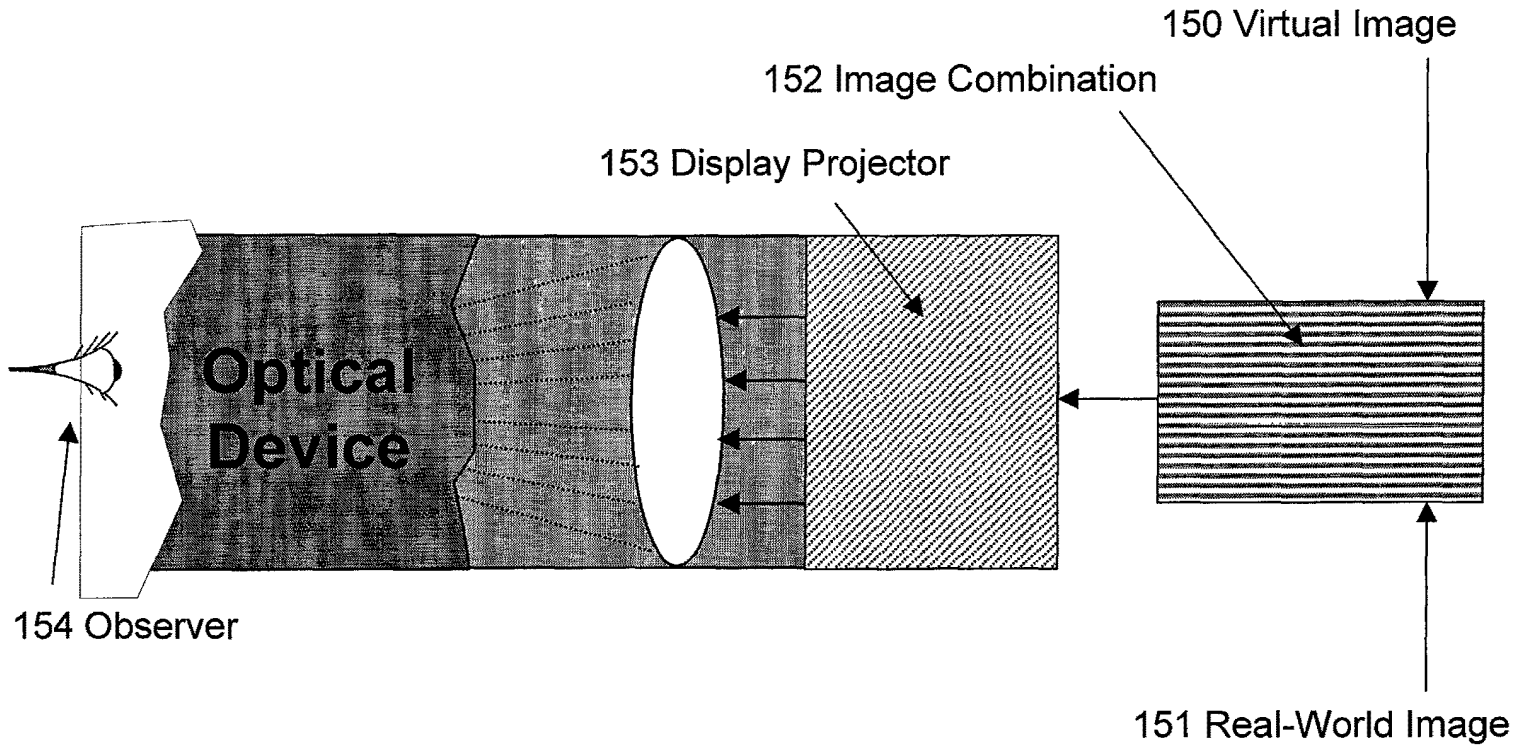


Figure 12: Combined Image Injection

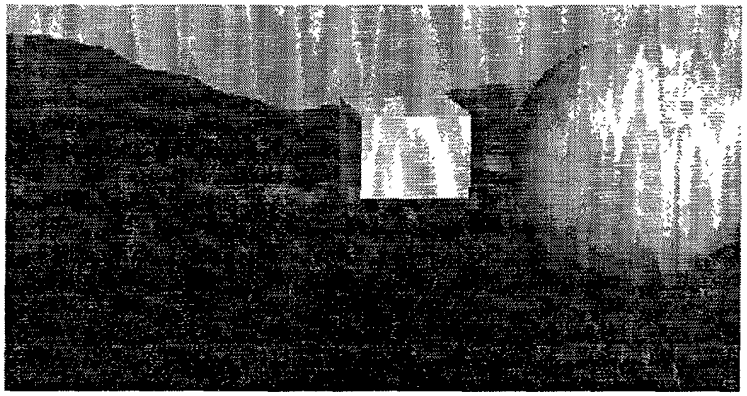


Figure 13: Effect of masking: the virtual images of Figure 6 in a real-world scene

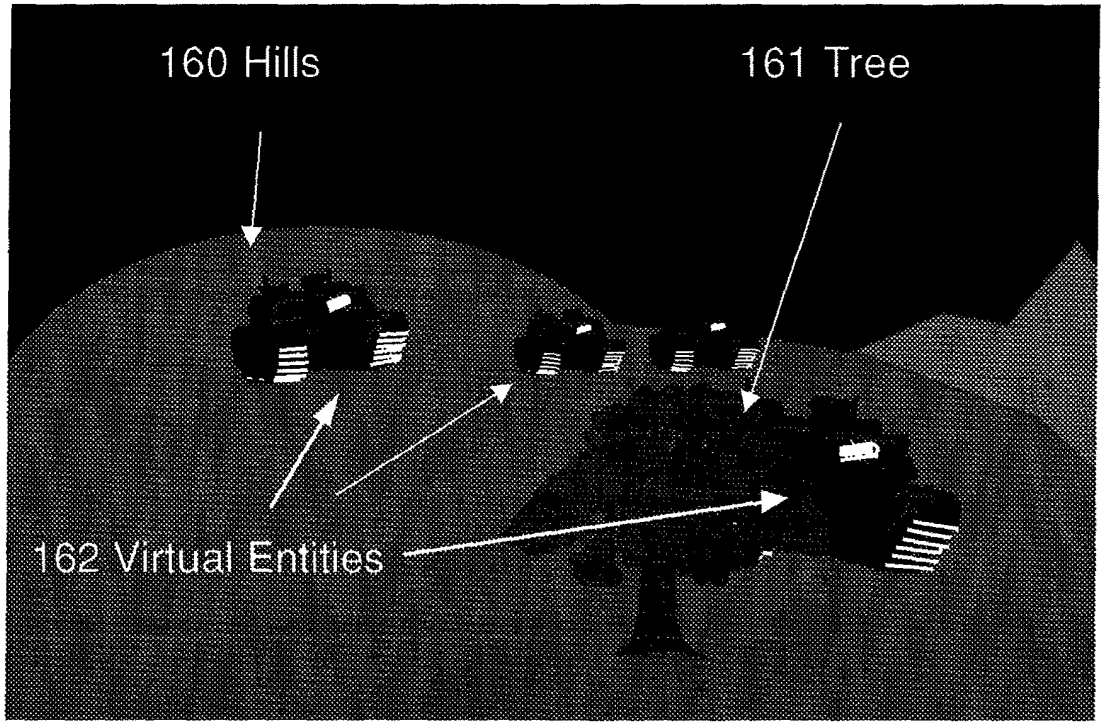


Figure 14: Combined Reality in Infrared

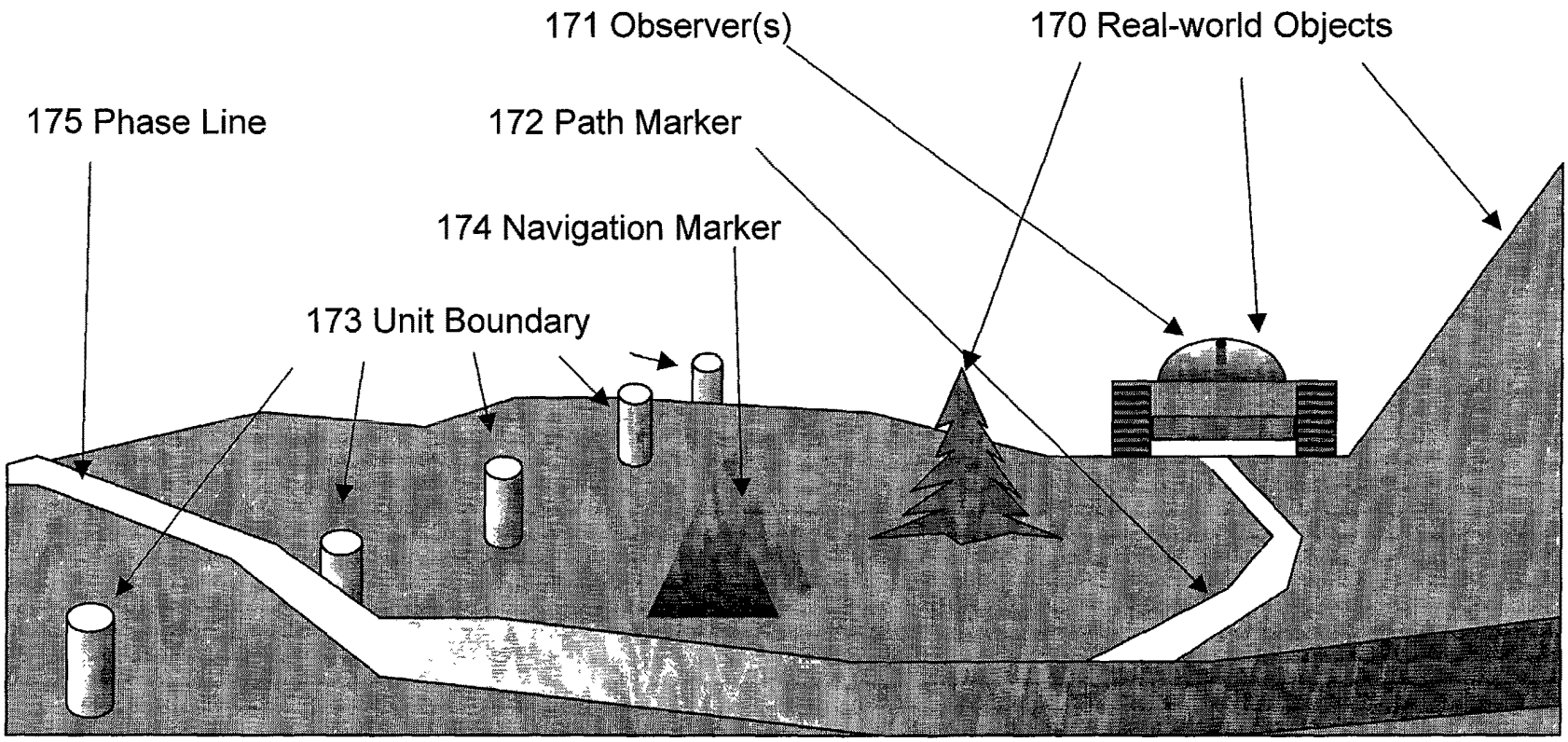


Figure 15: Conceptual Virtual Entities

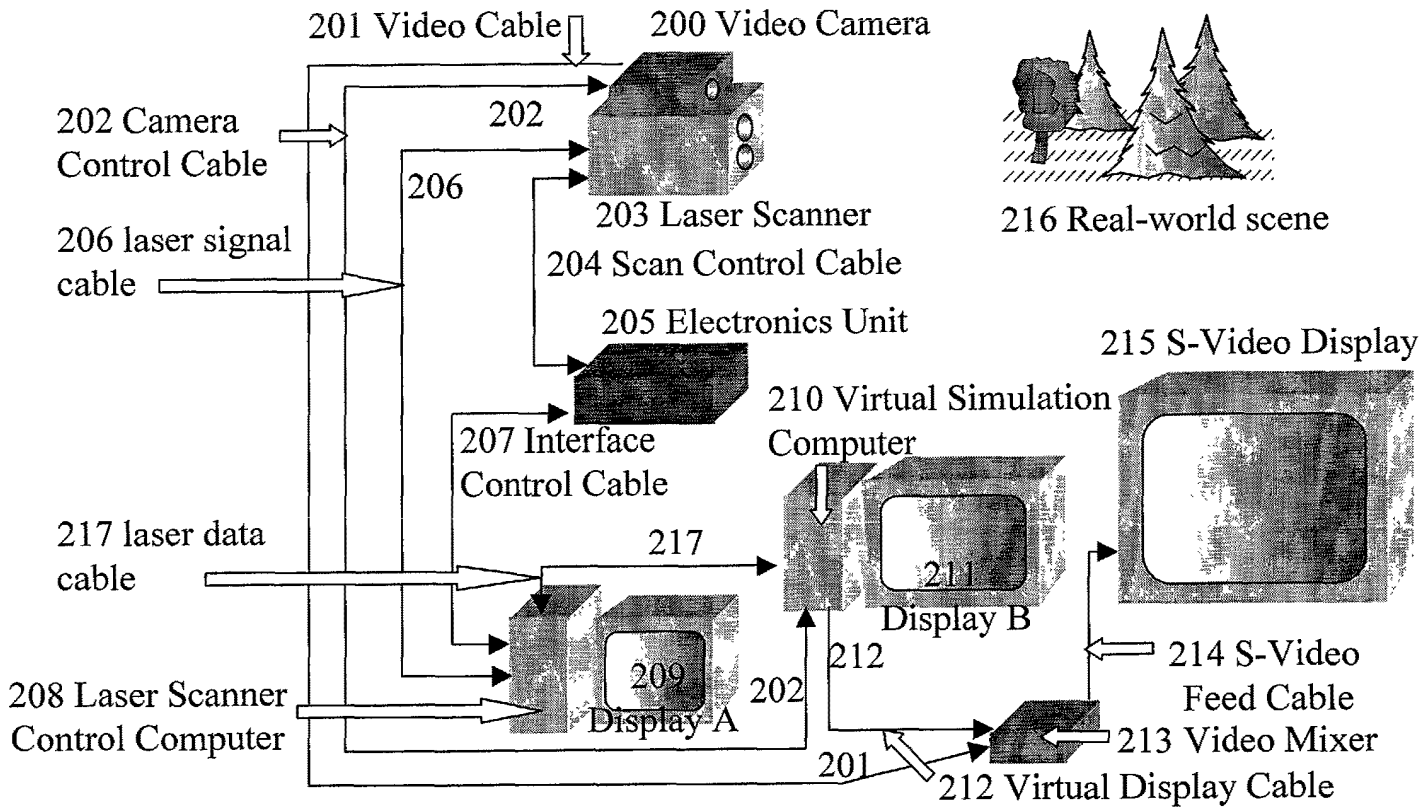

201 Video Cable
200 Video Camera
202 Camera Control Cable
202
206
203 Laser Scanner
216 Real-world scene
206 laser signal cable
204 Scan Control Cable
205 Electronics Unit
215 S-Video Display
210 Virtual Simulation Computer
207 Interface Control Cable
217 laser data cable
217
211
Display B
212
214 S-Video Feed Cable
208 Laser Scanner Control Computer
209
Display A
202
213 Video Mixer
201
212 Virtual Display Cable

Figure 16

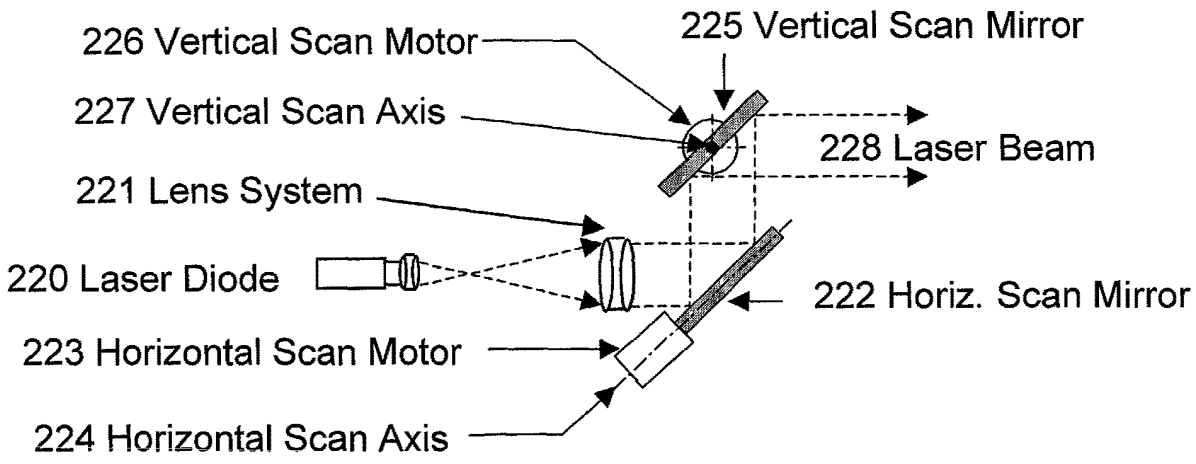


Figure 17: Laser Scanner Transmitter

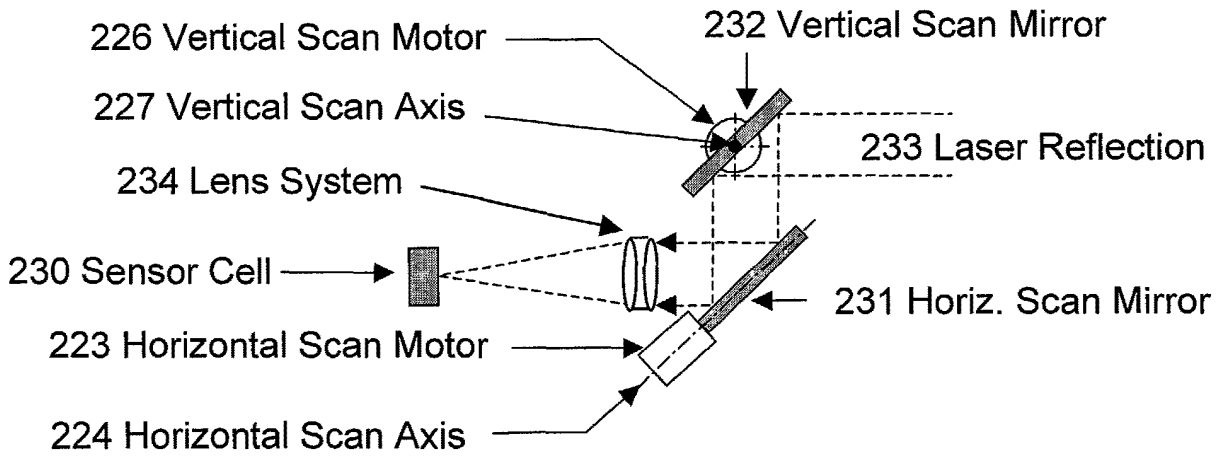


Figure 18: Laser Scanner Receiver

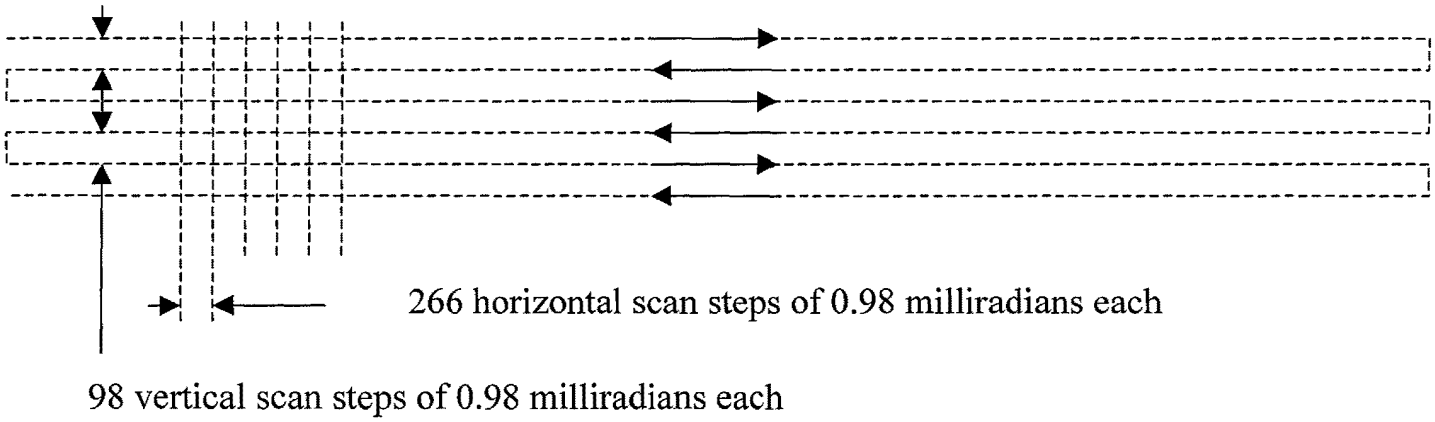


Figure 19: Laser scan pattern of the reference apparatus.

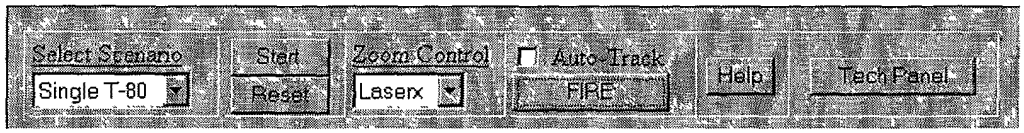


Figure 21: Control Frame in the Computer-generated Image

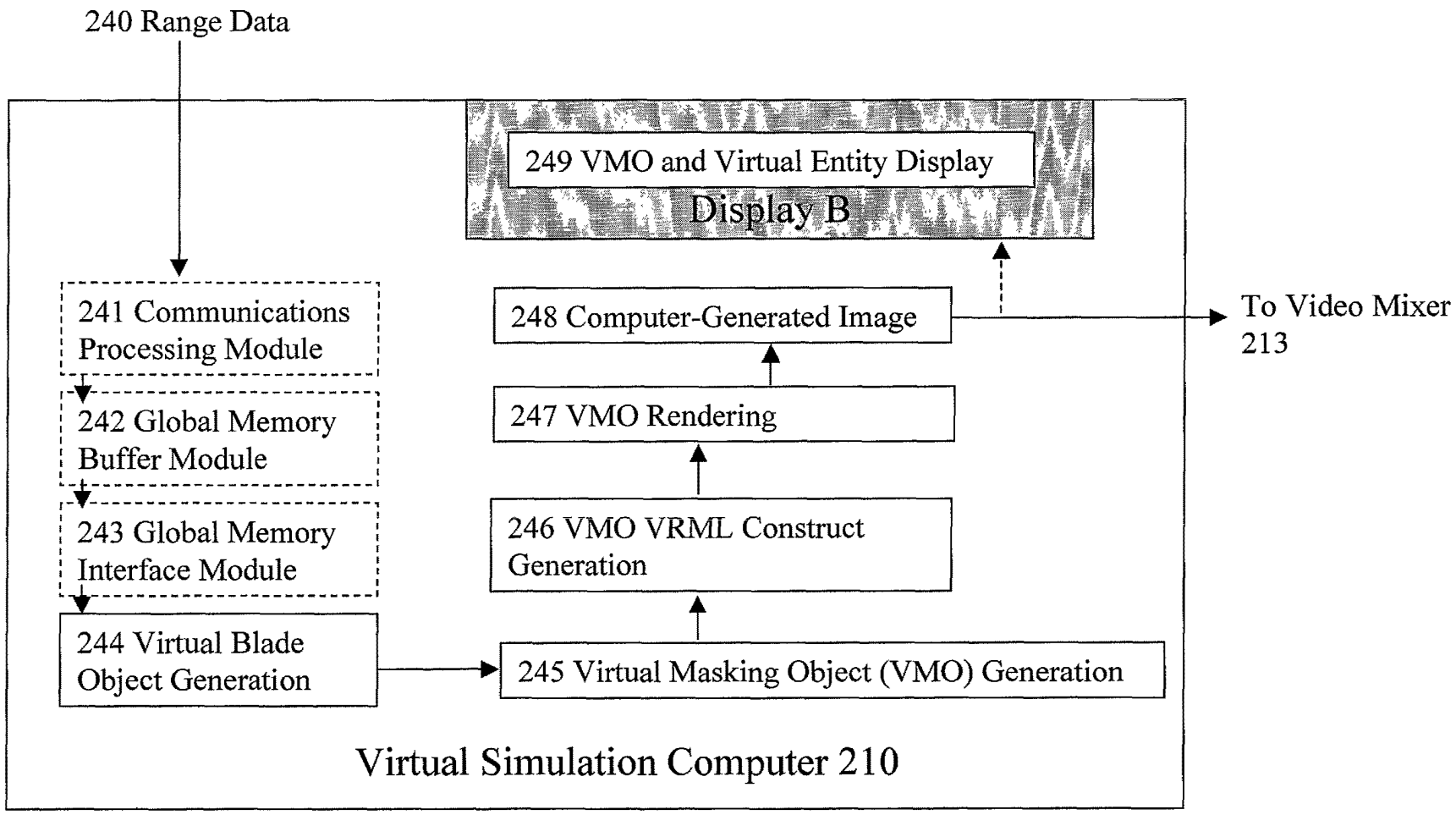


Figure 20: Processing Flow of Virtual Masking Object Generation

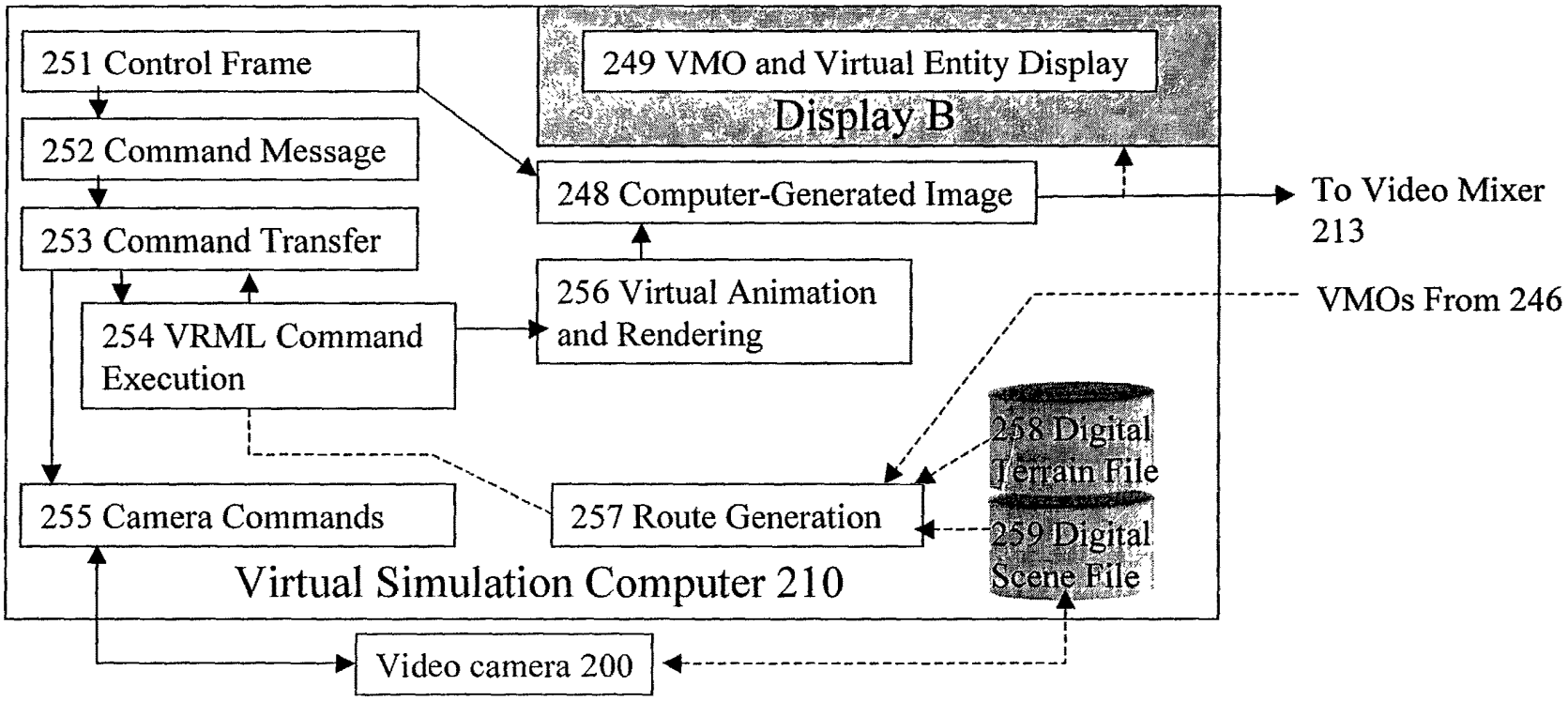


Figure 22: Processing Flow of the Virtual Simulation and Control Software

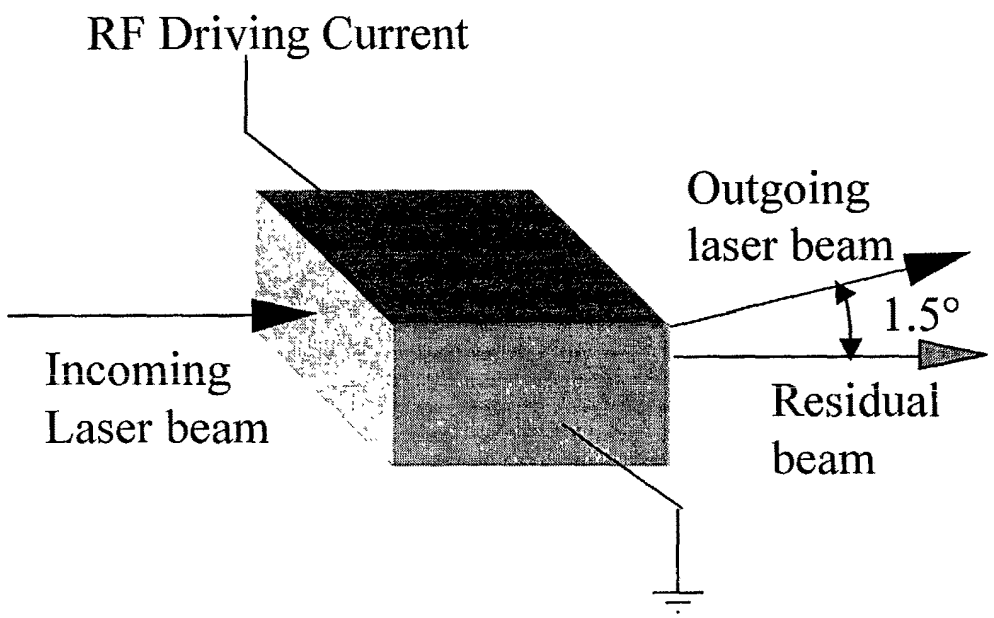


Figure 23: Crystal steering of laser beam

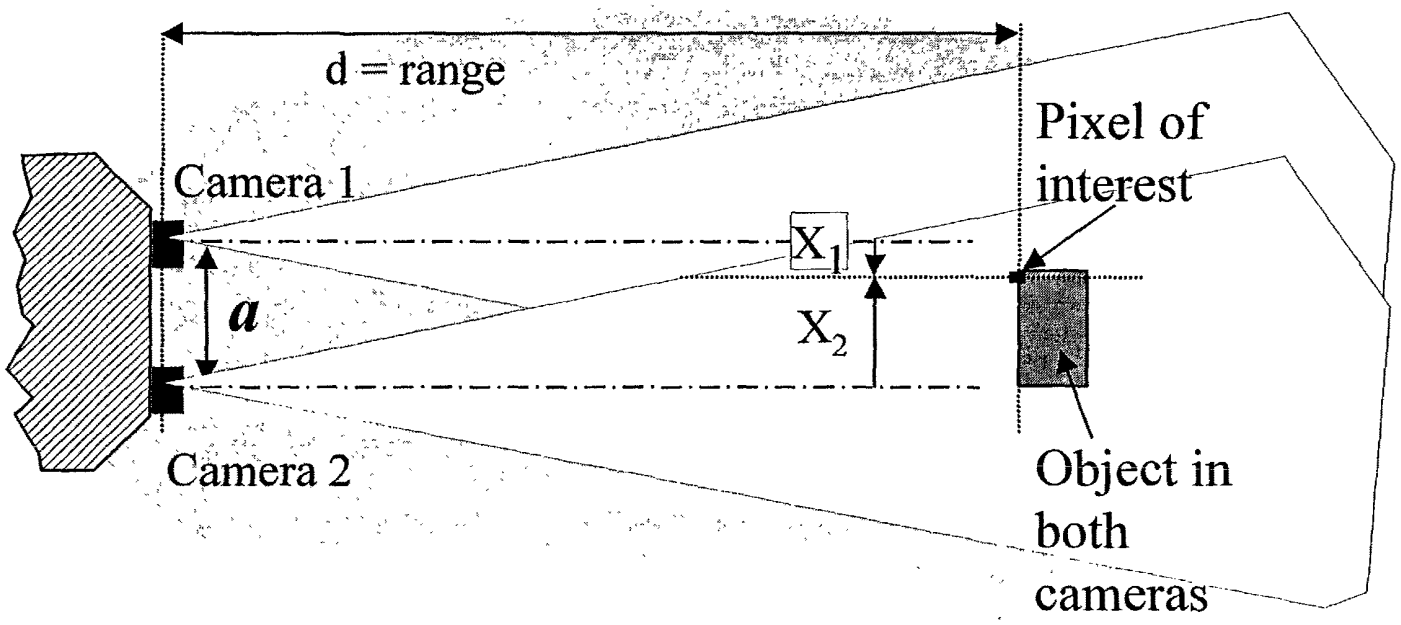


Figure 24: Passive or hybrid optical ranging

VERIFIED STATEMENT CLAIMING SMALL ENTITY STATUS (37 CFR 1.9(f) & [illegible])-SMALL BUSINESS CONCERN [illegible]	Docket No. (Optional) 8418/1

Applicant or Patentee: Ivan J. Jaszlics et al.
Serial or Patent No.:
Filed or Issued: Herewith
Title: System For Combining Virtual Images With Real-World Scenes

I hereby declare that I am

- ☐ the owner of the small business concern identified below:
- ☒ an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF SMALL BUSINESS CONCERN Pathfinder Systems, Inc.

ADDRESS OF SMALL BUSINESS CONCERN 274 Union Blvd, Suite 430
Lakewood, Colorado 80228

I hereby declare that the above identified small business concern qualifies as a small business concern as defined in 13 CFR 121.12, and reproduced in 37 CFR 1.9(d), for purposes of paying reduced fees to the United States Patent and Trademark Office, in that the number of employees of the concern, including those of its affiliates, does not exceed 500 persons. for purposes of this statement, (1`) the number of employees of the business concern is the average over the previous fiscal year of the concern of the persons employed on a full-time, part-time, or temporary basis during each of the pay periods of the fiscal year, and (2) concerns are affiliates of each other when either, directly or indirectly, one concern controls or has the power to control the other, or a third party or parties controls or has the power to control both.

I hereby declare that rights under contract law have been conveyed to and remain with the small business concern identified above with regard to the invention described in:

- ☒ the specification filed herewith with title as listed above.
- ☐ the application identified above.
- ☐ the patent identified above

If the rights held by the above identified small business concern are not exclusive, each individual, concern, or organization having rights in the invention must file separate verified statements averring to their status as small entities, and no rights to the inventions are held by any person, other than the inventor, who would not qualify as an independent inventor under 37 CFR 1.9(c) if that person made the invention, or by any concern which would not qualify as a small business concern under 27 CFR 1.9(d), or a nonprofit organization under 37 CFR 1.9(e).

Each person, concern or organization having any rights in the invention is listed below:

- ☒ no such person, concern, or organization exists.
- ☐ each such person, concern or organization is listed below.

Separate verified statements are required from each named person, concern, or organization having rights to the invention averring to their status as small entities. (37 CFR 1.27)

I acknowledge the duty to file, in this application or patent, notification of any change in status resulting in loss of entitlement to small entity status prior to paying, or at the time of paying, the earliest of the issue fee or any maintenance fee due after the date on which status as a small entity is no longer appropriate (37 CFR 1.28(b))

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under section 1001 of title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application, any patent issuing thereon, or any patent to which this verified statement is directed.

NAME OF PERSON SIGNING Ivan J. Jaszlics

TITLE OF PERSON IF OTHER THAN OWNER Vice President

ADDRESS OF PERSON SIGNING 274 Union Blvd, Suite 430, Lakewood, CO 80228

SIGNATURE: Ivan J. Jaszlics Date: November 24, 1997

(2-92) Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE (Reproduction)