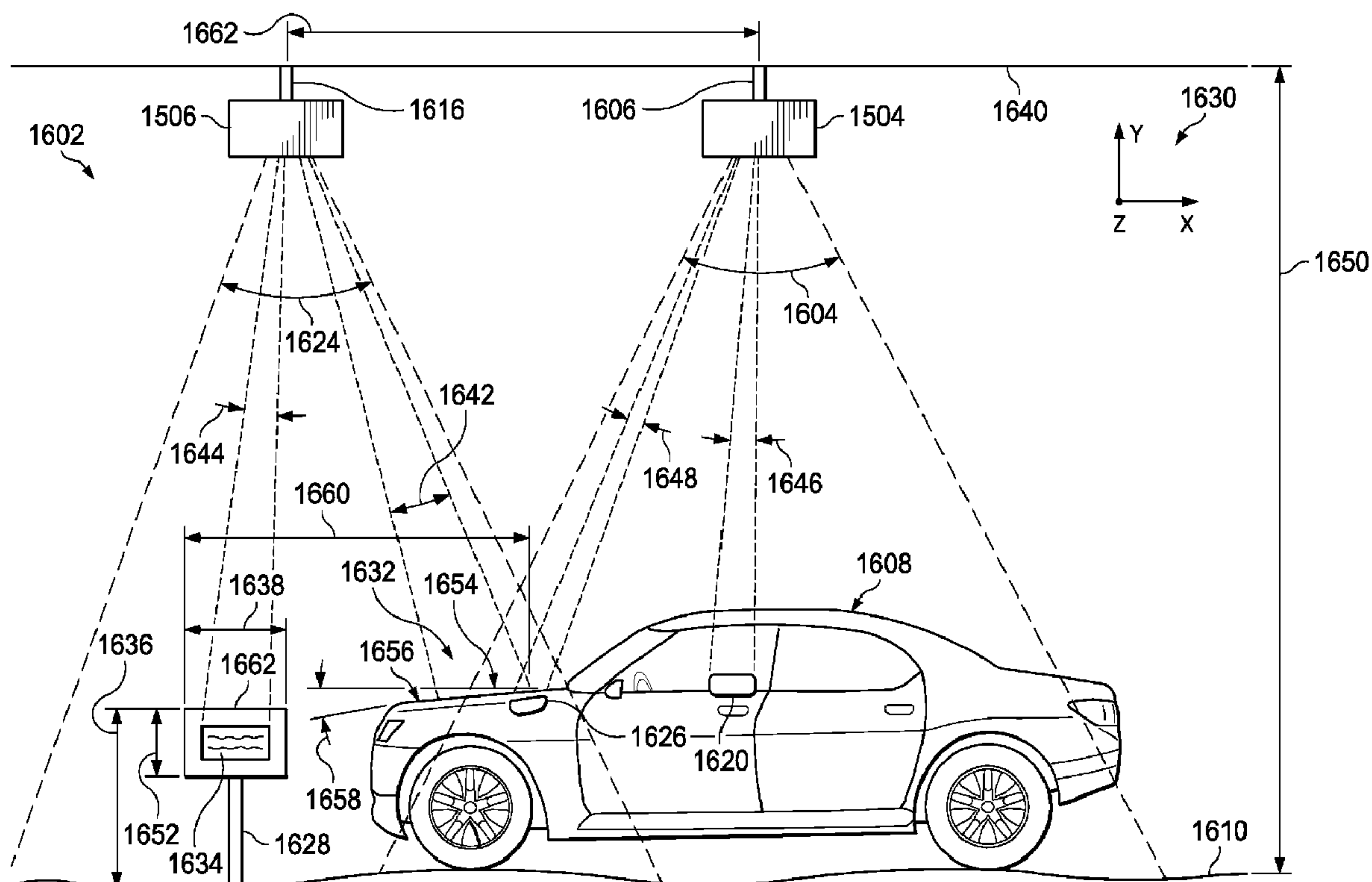




(86) Date de dépôt PCT/PCT Filing Date: 2015/12/30
(87) Date publication PCT/PCT Publication Date: 2016/07/07
(85) Entrée phase nationale/National Entry: 2017/06/15
(86) N° demande PCT/PCT Application No.: US 2015/068192
(87) N° publication PCT/PCT Publication No.: 2016/109749
(30) Priorité/Priority: 2014/12/30 (US62/097,769)

(51) Cl.Int./Int.Cl. *G03B 21/28* (2006.01),
G06K 9/62 (2006.01)
(71) Demandeur/Applicant:
OMNI CONSUMER PRODUCTS, LLC, US
(72) Inventeur/Inventor:
HOWARD, STEPHEN, US
(74) Agent: SIM & MCBURNEY

(54) Titre : SYSTEME ET PROCEDE DE PROJECTION INTERACTIVE
(54) Title: SYSTEM AND METHOD FOR INTERACTIVE PROJECTION



(57) **Abrégé(suite)/Abstract(continued):**

Specific content stored on the content server or locally in the memory of the computer processor is associated with each trigger area. A user interacts with the trigger areas and the system projects informational or entertainment content about the object on the surface of the object.

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

(19) World Intellectual Property
Organization
International Bureau

(43) International Publication Date
7 July 2016 (07.07.2016)



(10) International Publication Number
WO 2016/109749 A1

(51) International Patent Classification:
G03B 21/28 (2006.01)

(21) International Application Number:
PCT/US2015/068192

(22) International Filing Date:
30 December 2015 (30.12.2015)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
62/097,769 30 December 2014 (30.12.2014) US

(72) Inventor; and

(71) Applicant : **HOWARD, Stephen** [US/US]; P.O. Box 671046, Dallas, TX 75367 (US).

(74) Agent: **SCHULTZ, George, R.**; Schultz & Associates, P.C., 5400 LBJ Freeway, Suite 1200, Dallas, TX 75240 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM,

DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

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

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

(54) Title: SYSTEM AND METHOD FOR INTERACTIVE PROJECTION

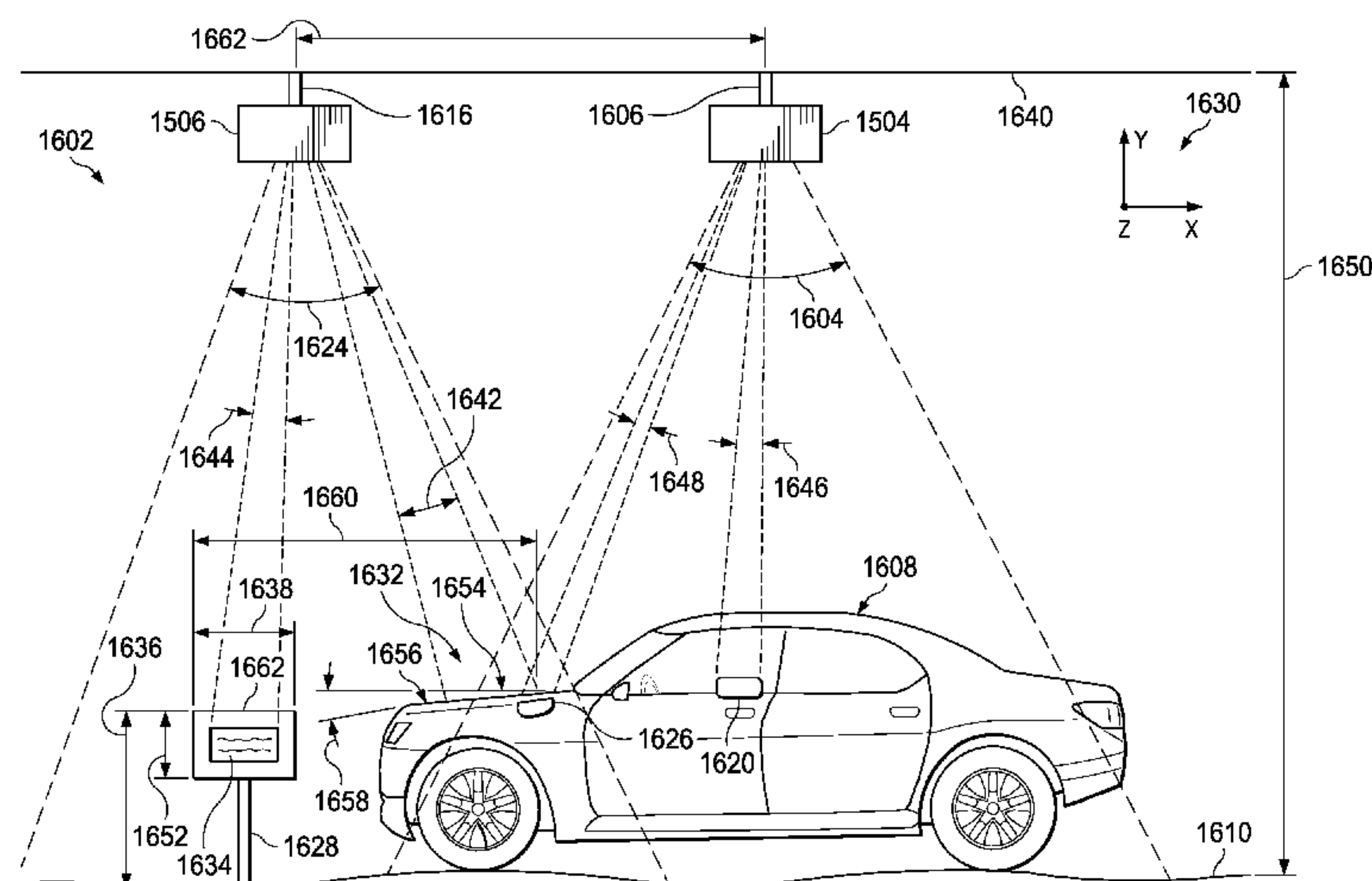


FIG. 16A

(57) Abstract: An interactive projection system and method comprising a camera/projector unit having a computer processor connected via a network to a content server and content database. The system projects interactive trigger areas on a three-dimensional object. Specific content stored on the content server or locally in the memory of the computer processor is associated with each trigger area. A user interacts with the trigger areas and the system projects informational or entertainment content about the object on the surface of the object.

WO 2016/109749 A1

SYSTEM AND METHOD FOR INTERACTIVE PROJECTION

FIELD OF THE INVENTION

[0001] The disclosure relates to systems and methods for projecting images. In particular, the disclosure relates to a system and method for projection of interactive areas and mapping of the interactive areas onto complex surfaces.

BACKGROUND OF THE INVENTION

[0002] Active product displays are useful in demonstrating advanced features of various products. Prior art systems are typically passive display systems, in that content is presented to a viewer on a screen with little opportunity for feedback. In other prior art systems, a touch screen computer is employed to allow the viewer to select content for review and to provide feedback. However, these systems require extensive custom set-up or are prohibitively expensive. Further, these systems require extensive maintenance due to frequent wear on the touch screen.

[0003] For example, U.S. Patent No. 7,598,942 to *Underkoffler, et al.* discloses a gestural interface to visually presented elements on a display screen. Multiple cameras capture movement and detect location and orientation and generate output signals to processors connected to a computer. The processors translate the camera outputs into gestural signals which are interpreted as input information. The computer uses the input information to generate commands to control computer functions.

[0004] U.S. Patent No. 7,034,807 to *Maggioni* discloses a system for interaction with a display. The system includes a recording unit for recording a pointer object in the vicinity of the display. A computer is connected to the recording unit and is used to determine a position of the pointer object. The system requires multiple cameras and multiple reflective surfaces.

[0005] U.S. Patent Publication No. 2012/0212413 to *Plagemann, et al.* discloses a system for receiving image information and translating it into computer instructions. Image information is received for a single predetermined action space to identify motion. A camera combined with mirrors, prisms, or optic cables is used to gather the image information. However, only one action area is monitored at any given time.

[0006] U.S. Patent Publication No. 2012/0162077 to *Sze, et al.* discloses an input device used to detect locations and motions of objects in a virtual working area. A camera is directed to a region of interest. A region of interest is illuminated by a “flat” beam of light. An object is placed in the region of interest and illuminated. The camera captures an image of the object and sends it to the processor. The processor processes the image to obtain locations and movements. Based on the movements, the processor produces computer instructions.

[0007] Therefore, there is a need for a system and method for projecting an interactive environment for a display. The system should provide options for the user to actively select the content to view. The system should be easy to set up and should be capable of being easily transported to other locales.

SUMMARY OF THE INVENTION

[0008] The system disclosed projects a “touchable” image on a three-dimensional object which enables user interaction. The touchable image is defined by a trigger area and a content area.

[0010] The system includes an infrared camera and a projector connected to a computer processor. In one embodiment, a content server and content database are also connected to the computer via a network. Content to be projected is stored in the database. During set-up, the system captures a baseline image of a three-dimensional target surface. The system then transforms the content to correct for projector position and for target curvature. During operation, the corrected content image is projected onto the three-dimensional target surface along with similarly transformed image of the trigger area. The infrared camera then monitors the trigger area for a trigger event and reports it to the server when it occurs. A trigger event occurs when an object is placed inside the trigger area for a predetermined period of time. Upon report of a trigger event, the system fetches the additional stored content, transforms it, and projects it onto the target surface. The computer processor uses both affine and a non-affine transforms to translate and “warp” the content image to match the surface of the target object.

[0011] In an alternate embodiment, the system uses facial recognition software to project predefined content associated with a particular recognized user on the surface of the object.

[0012] The action areas can be comprised of a plurality of separately defined actions areas each corresponding to different content. When a user interacts with an action area, the system recognizes the presence of a user and projects the associated content.

The affine transformations can be described through linear algebra. In linear algebra, linear transformations can be represented by matrices. If T is a linear transformation mapping $\mathbb{R}^n$ to $\mathbb{R}^m$ and $\vec{x}$ is a column vector with n entries, then $T(\vec{x}) = A\vec{x}$

[0013] For some $m \times n$ matrix A , called the transformation matrix of T . There are alternative expressions of transformation matrices involving row vectors that are preferred by some authors.

[0014] Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices).

[0015] Linear transformations are not the only ones that can be represented by matrices. Some transformations that are non-linear on a n -dimensional Euclidean space $\mathbb{R}^n$, can be represented as linear transformations on the $n+1$ -dimensional space $\mathbb{R}^{n+1}$. These include both affine transformations (such as translation) and projective transformations. For this reason, 4×4 transformation matrices are widely used in 3D computer graphics. These $n+1$ -dimensional transformation matrices are called, depending on their application, affine transformation matrices, projective transformation matrices, or more generally non-linear transformation matrices. With respect to an n -dimensional matrix, an $n+1$ -dimensional matrix can be described as an augmented matrix.

[0016] In the physical sciences, an active transformation is one which actually changes the physical position of a system, and makes sense even in the absence of a coordinate system whereas a passive transformation is a change in the coordinate description of the physical system (change of basis). The distinction between active and passive transformations is important. By default, by *transformation*, mathematicians usually mean active transformations,

while physicists could mean either.

[0017] Put differently, a *passive* transformation refers to description of the *same* object as viewed from two different coordinate frames.

[0018] If one has a linear transformation $T(x)$ in functional form, it is easy to determine the transformation matrix A by transforming each of the vectors of the standard basis by T , then inserting the result into the columns of a matrix. In other words,

$$A = [T(\vec{e}_1) \ T(\vec{e}_2) \ \dots \ T(\vec{e}_n)] \quad \text{Eq. 1}$$

[0019] For example, the function $T(x) = 5x$ is a linear transformation. Applying the above process (suppose that $n = 2$ in this case) reveals that

$$T(\vec{x}) = 5\vec{x} = 5\mathbf{I}\vec{x} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \vec{x} \quad \text{Eq. 2}$$

[0020] It must be noted that the matrix representation of vectors and operators depends on the chosen basis; a similar matrix will result from an alternate basis. Nevertheless, the method to find the components remains the same.

[0021] Mathematically, vector v can be represented in basis vectors, $E = [\vec{e}_1 \ \vec{e}_2 \ \dots \ \vec{e}_n]$ with coordinates $[v]_E = [v_1 \ v_2 \ \dots \ v_n]^T$:

$$\vec{v} = v_1 \vec{e}_1 + v_2 \vec{e}_2 + \dots + v_n \vec{e}_n = \sum v_i \vec{e}_i = E[v]_E \quad \text{Eq. 3}$$

[0022] Now, express the result of the transformation matrix A upon $\vec{v}$, in the given basis:

$$A(\vec{v}) = A(\sum v_i \vec{e}_i) = \sum v_i A(\vec{e}_i) = [A(\vec{e}_1) A(\vec{e}_2) \ \dots \ A(\vec{e}_n)][v]_E = \quad \text{Eq. 4}$$

$$= A \cdot [v]_E = [\vec{e}_1 \ \vec{e}_2 \ \dots \ \vec{e}_n] \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad \text{Eq. 4 (cont.)}$$

[0023] The $a_{i,j}$ elements of matrix A are determined for a given basis E by applying

A to every $\vec{e}_j = [00 \dots (v_j = 1) \dots 0]^T$, and observing the response vector $A\vec{e}_j = a_{1,j}\vec{e}_1 + a_{2,j}\vec{e}_2 + \dots + a_{n,j}\vec{e}_n = \sum a_{i,j} \vec{e}_i$. This equation defines the elements, $a_{i,j}$, of j-th column of the matrix A.

[0024] To represent affine transformations with matrices, we can use homogeneous coordinates. This means representing a 2-vector (x, y) as a 3-vector $(x, y, 1)$, and similarly for higher dimensions. Using this system, translation can be expressed with matrix multiplication. The functional form $x' = x + t_x; y' = y + t_y$ becomes:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{Eq. 5}$$

[0025] All ordinary linear transformations are included in the set of affine transformations, and can be described as a simplified form of affine transformations. Therefore, any linear transformation can also be represented by a general transformation matrix. The latter is obtained by expanding the corresponding linear transformation matrix by one row and column, filling the extra space with zeros except for the lower-right corner, which must be set to 1. For example, the clockwise rotation matrix from above becomes:

$$\begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 6}$$

[0026] Using transformation matrices containing homogeneous coordinates, translations can be seamlessly intermixed with all other types of transformations. The reason is that the real plane is mapped to the $w = 1$ plane in real projective space, and so translation in real Euclidean space can be represented as a shear in real projective space. Although a translation is a non-linear transformation in a 2-D or 3-D Euclidean space described by Cartesian coordinates, it becomes, in a 3-D or 4-D projective space described by homogeneous coordinates, a simple

linear transformation (a shear).

[0027] More affine transformations can be obtained by composition of two or more affine transformations. For example, given a translation T' with vector (t'_x, t'_y) , a rotation R by an angle θ counter-clockwise, a scaling S with factors (s_x, s_y) and a translation T of vector (t_x, t_y) , the result M of T'RST is:

$$\begin{bmatrix} s_x \cos \theta & -s_y \sin \theta & t_x s_x \cos \theta - t_y s_y \sin \theta + t'_x \\ s_x \sin \theta & s_y \cos \theta & t_x s_x \sin \theta + t_y s_y \cos \theta + t'_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Eq. 7}$$

[0028] When using affine transformations, the homogeneous component of a coordinate vector (normally called w) will never be altered. One can therefore safely assume that it is always 1 and ignore it. However, this is not true when using perspective projections.

[0029] Another type of transformation, of particular importance to this disclosure, is the perspective projection. Whereas parallel projections are used to project points onto the image plane along parallel lines, the perspective projection projects points onto the image plane along lines that emanate from a single point, called the center of projection. This means that an object has a smaller projection when it is far away from the center of projection and a larger projection when it is closer. This transformation is used to correct for the position of the projector relative to the three-dimensional target object.

[0030] The simplest perspective projection uses the origin as the center of projection, and $z = 1$ as the image plane. The functional form of this transformation is then $x' = x/z$; $y' = y/z$. . We can express this in homogeneous coordinates as:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \quad \text{Eq. 8}$$

[0031] After carrying out the matrix multiplication, the homogeneous component w_c will, in general, not be equal to 1. Therefore, to map back into the real plane we must perform the homogeneous divide or perspective divide by dividing each component by w_c :

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \frac{1}{w_c} \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} \quad \text{Eq. 9}$$

[0032] More complicated perspective projections can be composed by combining this one with rotations, scales, translations, and shears to move the image plane and center of projection wherever they are desired.

[0033] Non-affine transformations such as transformation of a two-dimensional surface to a three-dimensional surface can be accomplished, in theory, by treating the three-dimensional surface as a distortion or, in other words, a deviation from true rectilinear projection. This type of transformation is used to correct for the complex shape of the three-dimensional target object.

[0034] Although distortion can be irregular or follow many patterns, the most commonly encountered distortions are radially symmetric, or approximately so. Radial distortions can usually be classified as either “barrel” distortions or “pincushion” distortions.

[0035] In barrel distortion, image magnification decreases with distance from the optical axis. The apparent effect is that of an image which has been mapped around a sphere (or barrel). Fisheye lenses, which take hemispherical views, utilize this type of distortion as a way to map an infinitely wide object plane into a finite image area. In a zoom lens barrel distortion appears in the middle of the lens's focal length range and is worst at the wide-angle end of the range.

[0036] In pincushion distortion, image magnification increases with the distance from

the optical axis. The visible effect is that lines that do not go through the center of the image are bowed inwards, towards the center of the image, like a pincushion.

[0037] A mixture of both types, sometimes referred to as “mustache” distortion or complex distortion, is less common but not rare. It starts out as barrel distortion close to the image center and gradually turns into pincushion distortion towards the image periphery, making horizontal lines in the top half of the frame look like a handlebar mustache.

[0038] These distortions are radial distortions. They imply a nonlinear radius mapping from the object to the image. For example, what is seemingly pincushion distortion is simply an exaggerated radius mapping for large radii in comparison with small radii. Conversely, barrel distortion is simply a diminished radius mapping for large radii in comparison with small radii.

[0039] Radial distortion can be corrected using Brown's distortion model, also known as the Brown–Conrady model. The Brown–Conrady model corrects both for radial distortion and for tangential distortion caused by physical elements in a lens not being perfectly aligned. The latter is also known as decentering distortion.

$$x_d = x_u(1 + K_1r^2 + K_2r^4 + \dots) + (P_2(r^2 + 2x_u^2) + 2P_1x_uy_u)(1 + P_3r^2 + P_4r^4 + \dots) \quad \text{Eq. 10}$$

$$y_d = y_u(1 + K_1r^2 + K_2r^4 + \dots) + (P_1(r^2 + 2y_u^2) + 2P_2x_uy_u)(1 + P_3r^2 + P_4r^4 + \dots) \quad \text{Eq. 11}$$

where:

(x_d, y_d) = distorted image point as projected on image plane using specified lens,

(x_u, y_u) = undistorted image point as projected by an ideal pin-hole camera,

(x_c, y_c) = distortion center (assumed to be the principal point),

$K_n = n^{th}$ = radial distortion coefficient,

$P_n = n^{th}$ tangential distortion coefficient,

$r = \sqrt{(x_u - x_c)^2 + (y_u - y_c)^2}$, and

... = an infinite series.

[0040] In our case, barrel and pincushion distortions are quadratic, meaning they increase as the square of distance from the center. In mustache distortion the quartic (degree 4) term is significant: in the center, the degree 2 barrel distortion is dominant, while at the edge the degree 4 distortion in the pincushion direction dominates. Other distortions are in principle possible – pincushion in center and barrel at the edge, or higher order distortions (degree 6, degree 8) – but do not generally occur in practical cases, and higher order distortions are small relative to the main barrel and pincushion effects.

[0041] Barrel distortion typically will have a negative term for K_1 whereas pincushion distortion will have a positive value. Moustache distortion will have a non-monotonic radial geometric series where for some r the sequence will change sign.

BRIEF DESCRIPTION OF DRAWINGS

- [0042] Reference will now be made to the following drawings:
- [0043] Figure 1 is a system architecture diagram of a preferred embodiment.
- [0044] Figure 2A is a schematic of a camera/projection unit of a preferred embodiment.
- [0045] Figure 2B is a schematic of a camera/projection unit of a preferred embodiment.
- [0046] Figure 3 is a plan view of a display of a preferred embodiment.
- [0047] Figure 4 is a plan view of a display of a preferred embodiment.
- [0048] Figure 5 is a top view of a display of a preferred embodiment.
- [0049] Figure 6 is a plan view of a display of a preferred embodiment.
- [0050] Figure 7 is a plan view of a display of a preferred embodiment.
- [0051] Figure 8 is a plan view of a display of a preferred embodiment.
- [0052] Figure 9 is a plan view of a display of a preferred embodiment.
- [0053] Figure 10 is a plan view of a display of a preferred embodiment.
- [0054] Figure 11 is a flow chart of a calibration method of a preferred embodiment.
- [0055] Figure 12 is a flow chart of a run time process of a preferred embodiment.
- [0056] Figure 13 is a flow chart of a run time process for projection mapping of a preferred embodiment.
- [0057] Figure 14 is a flow chart of a facial recognition process of a preferred embodiment.
- [0058] Figure 15 is a system architecture diagram of a preferred embodiment.
- [0059] Figure 16A is an elevation view of a preferred embodiment.

[0060] Figure 16B is an elevation view of a preferred embodiment.

[0061] Figure 17 is an isometric view of trigger areas of a preferred embodiment.

[0062] Figure 18A is a diagram of a touchable area application in accordance with one embodiment of the disclosure.

[0063] Figure 18B is a diagram of a trigger data structure in accordance with one embodiment of the disclosure.

[0064] Figure 18C is a diagram of a content data structure in accordance with one embodiment of the disclosure.

[0065] Figure 19A is a data flow diagram of a system using trigger areas in accordance with one embodiment of the disclosure.

[0066] Figure 19B is a flow chart for updating a trigger area in accordance with one embodiment of the disclosure.

[0067] Figure 19C is a view of an area of a trigger area in accordance with one embodiment of the disclosure.

[0068] Figure 19D is a view of a volume of a trigger area in accordance with one embodiment of the disclosure.

[0069] Figure 19E is a flow chart for updating content in accordance with one embodiment of the disclosure.

[0070] Figure 19F shows the matrix transformations applied to content in accordance with one embodiment of the disclosure.

[0071] Figure 20A is a data flow diagram of a system using trigger areas in accordance with one embodiment of the disclosure.

[0072] Figure 20B is a flow chart for determining a touch of a trigger area in

accordance with one embodiment of the disclosure.

[0073] Figure 21A is a view of the application when a first tab is selected in accordance with one embodiment of the disclosure.

[0074] Figure 21B is a view of the application when a second tab is selected in accordance with one embodiment of the disclosure.

DETAILED DESCRIPTION

[0075] In the descriptions that follow, like parts are marked throughout the specification and drawings with the same numerals, respectively. The drawing figures are not necessarily drawn to scale and certain figures may be shown in exaggerated or generalized form in the interest of clarity and conciseness.

[0076] It will be appreciated by those skilled in the art that aspects of the present disclosure may be illustrated and described herein in any of a number of patentable classes or context including any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof. Therefore, aspects of the present disclosure may be implemented entirely in hardware or combining software and hardware implementation that may all generally be referred to herein as a “circuit,” “module,” “component,” or “system” (including firmware, resident software, micro-code, etc.). Further, aspects of the present disclosure may take the form of a computer program product embodied in one or more computer readable media having computer readable program code embodied thereon.

[0077] Any combination of one or more computer readable media may be utilized. The computer readable media may be a computer readable signal medium or a computer readable storage medium. For example, a computer readable storage medium may be, but not limited to, an electronic, magnetic, optical, electromagnetic, or semiconductor system, apparatus, or device, or any suitable combination of the foregoing. More specific examples of the computer readable storage medium would include, but are not limited to: a portable computer diskette, a hard disk, a random access memory (“RAM”), a read-only memory (“ROM”), an erasable programmable read-only memory (“EPROM” or Flash memory), an appropriate optical fiber with a repeater, a portable compact disc read-only memory (“CD-ROM”), an optical storage

device, a magnetic storage device, or any suitable combination of the foregoing. Thus, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

[0078] Computer program code for carrying out operations for aspects of the present disclosure may be written in any combination of one or more programming languages, markup languages, style sheets and JavaScript libraries, including but not limited to Windows Presentation Foundation (WPF), HTML/CSS, XAML, and JQuery.

[0079] Aspects of the present disclosure are described herein with reference to flowchart illustrations and/or block diagrams of methods, systems and computer program products according to embodiments of the disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a computer interconnected with dedicated and special purpose electronic devices, or other dedicated programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create an integrated machine to implement the functions described in the flowchart and the block diagrams.

[0080] These computer program instructions may also be stored in a computer readable medium that when executed can direct a computer, other programmable data processing apparatus, or other devices to function in a particular manner, such that the instructions when stored in the computer readable medium produce an article of manufacture including instructions which when executed, cause a computer to implement the function/act specified in the flowchart

and/or block diagram block or blocks. The computer program instructions may also be loaded onto a computer, other programmable instruction execution apparatus, or other devices to cause a series of operational steps to be performed on the computer, other programmable apparatuses or other devices to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide processes for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0081] Referring to Figure 1, system **100** includes network **101**, content server **102** connected to network **101**, and camera/projector unit **104** connected to network **101**. Content server **102** is further connected to content database **103** for the storage of multimedia and other relevant data.

[0082] In a preferred embodiment, network **101** is a wide area network such as the internet, but can include other wide area and intranet networks.

[0083] In a preferred embodiment, camera/projector unit **104** downloads content stored in content database **103** from content server **102** and saved to local memory for later projection. In another embodiment, camera/projector unit **104** streams content stored in content database **103** from content server **102** for projection.

[0084] Referring to Figure 2A in one embodiment, camera/projector unit **200** includes enclosure **201**, computer/processor **202**, projector **203** connected to computer/processor **202**, and camera **204** connected to computer/processor **202**.

[0085] Referring to Figure 2B in another embodiment, camera/projector unit **205** positions mirror **206** at angle α adjacent to projector **203** to redirect the field of view of projector **203**.

[0086] In a preferred embodiment, computer/processor **202** is a laptop computer

having a suitable processor and memory for storage of content and database, and any other relevant data. In another embodiment, computer/processor **202** is a tablet computer.

[0087] A projection application is stored in the memory of computer/processor **202** for defining and projecting a set of trigger areas with projector **203**, detecting a user presence and recognizing the user with camera **204**, detecting a selection of the set of trigger areas with camera **204**, and projecting content with projector **203**, as will be further described below.

[0088] In a preferred embodiment, camera **204** is a 3-D range camera, such as the Kinect camera from Microsoft Corporation. Camera **204** provides image data in the form of streaming video at the rate of 30 frames per second in a standard 640 x 480 VGA. The focal length of the camera is 525 pixels with a field of view of approximately 63°. Camera **204** also provides distance-to-target information from an on-board infrared transmitter/receiver. The focal length of the infrared transmitter/receiver is 580 pixels with a field of view of approximately 58°. The infrared data is provided as a 16 bit number at a refresh rate of 200 μ s.

[0089] Referring to Figure 3, display **300** includes camera/projector unit **301** having field of view **302**. Camera/ projector unit **301** is connected to arm **304**. Arm **304** is attached to and extends from an overhead ceiling. In a preferred embodiment, field of view **302** can range from approximately 45° to 75°. Product **303** is positioned in field of view **302**.

[0090] Referring to Figure 4 in another embodiment, display **400** includes camera/projector unit **401** having field of view **403**. Camera/projector unit **401** is connected to and supported by stand **402**. Stand **402** is removably connected to the floor. Product **404** is positioned in field of view **403**.

[0091] Referring to Figure 5, display **500** includes camera/projector unit **501** attached to arm **502** which is attached to base **508**. Camera/projector unit **501** projects trigger areas **504**

and **505** onto the surface of product **503**. When a user touches the surface of product **503** in trigger areas **504** and **505**, camera/projector unit **501** detects the presence of the user in trigger areas **504** and **505** and projects contents **506** and **507**, respectively, onto the surface of product **503**. In a preferred embodiment, each of contents **506** and **507** is a video. Any type of graphical images may be employed. For example, if a user touches the hood of the car shown in the drawing, an image of a running engine taken from the same perspective as the user may be projected on the hood to simulate a “see through” effect to alert the user to “see” the running engine through the hood.

[0092] Referring to Figure 6 in another embodiment, display **600** includes camera/projector unit **601** having field of view **602** covering product **603**. Camera/projector unit **601** projects trigger areas **604**, **605**, **606**, and **607** onto the surface of product **603**. When a user touches the surface of product in trigger areas **604**, **605**, **606**, and **607**, camera/projector unit **601** projects content **615** onto the surface of product **603**.

[0093] In one embodiment, if a user touches the surface of product **603** in any of the trigger areas **604**, **605**, **606**, and **607**, camera/projector unit **601** projects a different color/pattern **616** onto product **603**, thereby enabling the user to visualize a potential change to the color or pattern of product **603**.

[0094] Stand **608** is positioned adjacent to product **603** and is covered by field of view **602**. Stand **608** includes placard **609**, support **610** attached to placard **609**, and base **611** attached to support **610**. Base **611** is removably attached to the floor. Camera/projector unit **601** projects trigger areas **612** and **613** onto the surface of placard **609**. When a user touches the surface of placard **609** in trigger areas **612** and **613**, camera/projector unit **601** projects content **614** onto placard **609**, thereby transforming placard **609** into an interactive digital sign. Content

614 preferably includes information, videos, and/or pictures of product **603**. Content **614** is retrieved from the content server or from the local memory of camera/projector unit **601**.

[0095] Referring to Figure 7, display **700** includes camera/projector units **701** and **702**. Camera/projector unit **701** has field of view **704** covering product **703** including product side **706**. Camera/projector unit **702** has field of view **705** covering product **703** including product side **707**. Camera/projector unit **701** projects trigger area **708** onto the surface of product side **706**. When a user touches the surface of product side **706** in trigger area **708**, camera/projector unit **701** projects a video or graphics content onto the surface of product side **706**. Camera/projector unit **702** projects trigger area **709** onto the surface of product side **707**. When the user touches the surface of product side **707** in trigger area **709**, camera/projector unit **702** projects a video or graphics content onto the surface of product side **707**.

[0096] Referring to Figure 8 in another embodiment, display **800** includes product **802** positioned in view of camera/projector unit **801**. Camera/projector unit **801** detects the position and presence of user **803** on product **802**. In one embodiment, an identity of user **803** is determined through a facial recognition process, as will be further described below. Trigger areas **805** and **806** are projected by camera/projector **801** onto the surface of product **802** adjacent to user **803** in response to the presence of user **803**.

[0097] In one embodiment, content **804** is automatically projected onto the surface of product **802** in response to the presence and identity of user **803**.

[0098] In another embodiment, content **804** is projected onto the surface of product **802** in response to user **803** touching any of trigger areas **805** and **806**.

[0099] In another embodiment, camera/projector unit **801** projects a first video as content **804** onto the surface of product **802** based on the detected presence of user **803**. When

user **803** touches any of trigger areas **805** and **806**, content **804** changes to a second video and a third video, respectively. For example, when camera/projector unit **801** detects the presence of user **803** on product **802**, content **804** is an introduction or welcome video. When user **803** touches trigger area **805**, content **804** is switched to a graphical brochure. When user **803** touches trigger area **806**, content **804** is switched to a product animation. Any type of content may be employed for content **804**.

[0100] Referring to Figure 9 in another embodiment, display **900** includes camera/projector **901** and product **902**. In this embodiment, projected content **903** includes and sequentially displays the components of product **902** including base layer **904**, support layer **905**, padding layer **906** and top layer **907**. In a preferred embodiment, projected content **903** includes a set of animations which represent a “peeling away” of each layer. For example, in response to a user interacting with a trigger area, top layer **907** is “peeled away” to reveal padding layer **906**. Each layer in succession can be projected individually on product **902** thus enabling the user to visualize the different layers of the product. In one embodiment, projected content **903** covers the entirety of product **902**. In another embodiment, projected content **903** covers a portion of product **902**.

[0101] Referring to Figure 10 in another embodiment, display **1000** includes camera/projector **1001** and product **1002**. Product **1002** rests on floor **1004**. Content **1003** is projected by camera/projector unit **1001** onto floor **1004** surrounding product **1002**. Content **1003** includes animation which gives the appearance that product **1002** is driving through various environments, such as rain, snow, or mud to give the appearance that product **1002** is moving.

[0102] Referring to Figure 11, method **1100** for calibrating camera/projector unit **200**

will be described. At step **1101**, a baseline image of a surface is determined. In this step, the infrared transmitter/receiver of camera **204** captures a baseline image including depth and color data of the surface of a product. In one embodiment, a thin cover, such as a sheet may be employed to cover a highly reflective surface, such as a surface of a car in order to capture an accurate baseline image.

[0103] At step **1102**, a position for each of a set of trigger areas for the product is selected. In a preferred embodiment, each trigger area is an ellipse. Other shapes may be employed. At step **1103**, the set of trigger areas is projected on the surface. At step **1104**, each trigger area is calibrated by placing a three-dimensional object on the trigger area, such as a hemispherically shaped object. Other three-dimensional objects may be employed. In this step, a position of the three-dimensional object relative to the saved baseline image is captured with camera/projector unit **200**.

[0104] At step **1105**, the captured position of each three-dimensional object is saved in the memory of camera/projector unit **200**. At step **1106**, a polygon is warped to match the surface, thereby ensuring that the polygon conforms to the surface on which it will be projected. The polygon may correspond to the entire surface or a section of the surface.

[0105] In a preferred embodiment, a non-affine transform is employed to map a set of points of the polygon to the surface. For example, if the polygon is a square in a flat plane and the surface is an angled surface with respect to the projector, then the points of the square are mapped to a quadrilateral that distorts the square image so that when projected on the angled surface, the projected quadrilateral appears as a square on the angled surface. Other transforms, mapping, and distortion techniques may be employed.

[0106] At step **1107**, the warped polygon is associated with the trigger area. At step

1108, a set of content is determined for the polygon. In this step, a file location of the desired set of content is associated with the polygon. The set of content is downloaded and saved to local memory or linked to a stream for the set of content. At step **1109**, the set of content is warped to match the warped polygon so that the selection of the trigger area projects the desired content undistorted on the surface.

[0107] Referring to Figure 12 in one embodiment, runtime process **1200** will be described. At step **1201**, runtime process **1200** begins. At step **1202**, a set of trigger areas is projected onto a surface. At step **1203**, a touch event is determined, i.e., a determination is made whether a user is in any of the set of trigger areas. In this step, the camera is queried for a user position. The user position is compared to the coordinates of a boundary for the trigger area. If the user position is not within the boundary of the trigger area, then runtime process returns to step **1202**. If the user position is within the boundary of the trigger area, then runtime process proceeds to step **1204**.

[0108] At step **1204**, a polygon with a set of content associated with the trigger area is projected onto the surface. At step **1205**, the presence of an end event is determined. An end event is a predetermined time period or another touch event by the user in the set of trigger areas. The predetermined time period is the runtime of the set of content or any desired time period. If an end event has not occurred, then runtime process **1200** returns to step **1204** to continue projecting the polygon and the set of content. If an end event has occurred, then runtime process **1200** proceeds to step **1206**.

[0109] At step **1206**, whether a terminate command has been received is determined. If a terminate command has not been received, then runtime process **1200** returns to step **1202**. If a terminate command has been received, then runtime process **1200** ends at step **1207**.

[0110] Referring to Figure 13 in another embodiment, runtime process **1300** for projection mapping will be described. At step **1301**, a scan of an area, such as a room is performed by a camera to detect any surfaces. At step **1302**, the scan of a detected surface is mapped to a virtual area. At step **1303**, a polygon and a set of content is warped to match the virtual area.

[0111] In a preferred embodiment, a non-affine transform is employed to map a set of points of the polygon and the set of content to the surface. In other embodiments, other transforms, mapping, and distortion techniques may be employed.

[0112] At step **1304**, the camera is queried to determine a presence of a user in the area. At step **1305**, if a presence is not detected, then runtime process **1300** returns to step **1304**. If a presence is detected, then a set of trigger areas are projected onto the surface at step **1306**. At step **1307**, a facial recognition process begins to recognize an identity of the user, as will be further described below.

[0113] At step **1308**, if the identity of the user is recognized then runtime process **1300** proceeds to step **1309**. At step **1309**, the polygon and a set of content associated with the user is projected onto the surface. At step **1311**, the presence of an end event is determined. An end event is a predetermined time period or another touch event by the user in the set of trigger areas. The predetermined time period is the runtime of the content or any desired time period. If an end event has not occurred, then runtime process **1300** returns to step **1309** to continue projecting the polygon and the set of content. If an end event has occurred, then runtime process **1300** proceeds to step **1313**.

[0114] If the identity of the user is not recognized in step **1308**, then runtime process **1300** proceeds to step **1310**. At step **1310**, the polygon and a set of predetermined content is

projected onto the surface.

[0115] At step **1312**, the presence of an end event is determined. If an end event has not occurred, then runtime process **1300** returns to step **1310** to continue projecting the polygon and the set of predetermined content. If an end event has occurred, then runtime process **1300** proceeds to step **1313**.

[0116] At step **1313**, a touch event determination is made as to whether a user is in the set of trigger areas. In this step, the camera is queried for a user position. The user position is compared to the coordinates of a boundary for any of the set of trigger areas. If the position is not within the boundary of any of the set of the trigger areas, then runtime process **1300** returns proceeds to step **1308**. If the position is within the boundary of any of the set of trigger areas, then runtime process proceeds to step **1314**.

[0117] At step **1314**, a polygon with a set of content associated with the trigger area is projected onto the surface. At step **1315**, the presence of an end event is determined. If an end event has not occurred, then runtime process **1300** returns to step **1314** to continue projecting the polygon and the set of content. If an end event has occurred, then runtime process **1300** proceeds to step **1316**.

[0118] At step **1316**, whether a terminate command has been received is determined. If a terminate command has not been received, then runtime process **1300** returns to step **1304**. If a terminate command has been received, then runtime process **1300** ends at step **1317**.

[0119] Referring to Figure 14, step **1307** will be further described as facial recognition process **1400**. Facial recognition process begins at step **1401**. At step **1402**, a camera is queried to record the locations of facial features including eye location and color, ear location, chin location and hair color. In a preferred embodiment, the query is accommodated by

Microsoft SKD Rel. 7.1 .LIB file Microsoft.Kinect.Toolkit.FaceTracking. Any type of facial tracking may be employed. In a preferred embodiment, at least 121 different locations are determined. At step **1403**, the distances between facial features are measured. At step **1404**, a “facial map” is created. The facial map includes a table of the facial features and the distances between them. At step **1405**, the facial map is compared to a set of digital images stored in memory. A result of the comparison is output at step **1406**. At step **1407**, the process ends.

[0120] Referring to Figure 15, in one embodiment, components of interactive projection system **1500** are local computer **1502**, camera **1504**, and projector **1506**. Local computer **1502**, camera **1504**, and projector **1506** can be separate components positioned individually or all be contained in a single housing. Camera **1504** collects color and depth data within its field of view, constantly monitoring defined trigger areas for interaction. Projector **1506** displays projectable content and trigger areas on any surface within its field of view. Local computer **1502** stores and manipulates camera data, content files, locations of projected content, and locations of trigger areas and projected trigger areas. Local computer **1502** analyzes camera data to determine if interaction with any trigger areas has occurred as will be further described. Local computer **1502** includes a transform function capable of warping content images in order to project the content on three-dimensional surfaces undistorted as will be further described.

[0121] In a preferred embodiment, local computer **1502** contains a processor; a memory for storage of content, a database, and any other relevant data; and a software module including hardware drivers. Minimum requirements for local computer **1502** include an Intel Core I7 processor, at least eight (8) GB of RAM, at least five (5) GB of local memory, a 2 GB graphics processor, Windows operating system 8.1 or higher, and a USB3 connection to camera **1504**. An example of a suitable computer is the Latitude E6540 laptop computer available from

Dell. In a preferred embodiment, hardware drivers include an image capturing program such as Kinect for Windows available from Microsoft.

[0122] Local computer **1502** is connected to camera **1504**. Camera **1504** is a 3-D range camera. Camera **1504** includes firmware for movement interpretation and image recognition. Camera **1504** provides image color data in the form of streaming video at the rate of 30 frames per second in a standard 640 X 480 VGA format at 4 bytes/pixel. The focal length of camera **1504** is 525 pixels with a field of view that can range between approximately 45° to 75°. Camera **1504** also provides distance to target information from an on-board infrared transmitter/receiver pair. The focal length of the infrared transmitter/receiver is 580 pixels with a field of view in the range of approximately 45° to 75°. The infrared data is provided as a 16 bit number at a refresh rate of 200 μ s. An example of a suitable camera is the Kinect camera available from Microsoft.

[0123] Local computer **1502** is connected to projector **1506** through an HDMI port and cable. Projector **1506** is a PC ready projector having a brightness of 10,000 lumens and a resolution of 1920 X 1200 WUXGA. Projector **1506** has a throw distance in the range of 4 feet to 40 feet with a standard lens. Alternate lenses are available which can dramatically increase the throw distance of projector **1506**. Projector **1506** has a field of view angle in the range of 45° to 75° but can be adjusted with different lenses. An example of a suitable projector is the PLWU8100F available from ViviBright.

[0124] Local computer **1502** is connected to microphone **1512** and speaker **1514**. Microphone **1512** allow for the reception of audible commands that can be used to trigger the display or playback of images, video, and audio with respect to one or more content areas. Speaker **1514** allows for the playback of the audio channels associated with a video and for the

playback of audio files that have been associated with a trigger area.

[0125] In an alternate embodiment, components of interactive projection system **1500** further include network **1510**, content server **1522**, database **1524**, and control computer **1520**. Content server **1522** is connected to network **1510**. Network **1510** connects local computer **1502** to content server **1522**. Content server **1522** is further connected to database **1524** for the storage of multimedia and other relevant data. Network **1510** is a wide area network such as the Internet but can include other wide area or intranet networks. Local computer **1502** can download content stored in database **1524** from content server **1522** and save the content to local memory for later projection. Local computer **1502** can also stream content stored in database **1524** from content server **1522** for projection with projector **1506**.

[0126] Control computer **1520** is connected to content server **1522** and connected to local computer **1502** through network **1510**. Through control computer **1520**, a user can manipulate local computer **1502** and set up interactive projection system **1500** from a remote location. Minimum requirements for control computer **1520** include an Intel Core I7 processor, at least eight (8) GB of RAM, at least five (5) GB of local memory, a 2 GB graphics processor, and Windows operating system 8.1 or higher.

[0127] Referring to Figure 16A and coordinate system **1630**, the *X* axis represents horizontal, the *Y* axis represents vertical, and the *Z* axis is out of the page. The relative positions of the components within a viewing space **1602** are shown. Viewing space **1602** is a room having a light level sufficient for the projection of projector **1506** to be adequately viewed by a user. The dimensions of viewing space **1602** are those of a typical showroom floor but can vary greatly dependent on the size of the object of the interactive display and the lens of the projector. In general, camera **1504** is positioned to achieve field of view **1604**. Field of view **1604** can

range from 45° - 75°. Camera **1504** is suspended from ceiling **1640** with arm **1606** but can also be attached to a floor stand with equal success. Object **1608** is positioned on floor **1610** within field of view **1604**. The locations of any trigger areas must be within field of view **1604**.

[0128] In general, projector **1506** is positioned to achieve field of view **1624**. Field of view **1624** can range from 45° - 75°. Projector **1506** may be suspended from ceiling **1640** with arm **1616** but can also extend from a floor stand. Object **1608** is positioned on floor **1610** within field of view **1624**. The location of any projected content must be within field of view **1604**. Projector **1506** and camera **1504** are separated by a distance **1662** along the *X* axis and by distance **1664** along the *Y* axis.

[0129] Placard **1622** extends from floor **1610** via stand **1628**. As shown, content may be displayed on placard **1622** because placard **1622** is positioned within field of view **1624** of projector **1506**. In another embodiment, placard **1622** is positioned within field of view **1604** of camera **1504**. In another embodiment, placard **1622** is in field of view **1624** of projector **1506** and field of view **1604** of camera **1504** and can display both content areas and trigger areas. In a preferred embodiment, the total height **1636** of the stand and placard is about 3 to 4 feet, the length **1638** of the placard is about 16 inches, and the width **1652** of the placard is about 12 inches. Distance **1650** is the distance between ceiling **1640** and floor **1610** and is about 16 feet in one preferred embodiment.

[0130] Field of view **1624** of projector **1506** is about 72 degrees and includes field of view **1642** and field of view **1644**. Field of view **1642** is the field of view for content area **1632** on the surface of object **1608** and is about 7 degrees. Field of view **1644** is the field of view for content area **1634** on placard **1622** and is about 5 degrees. Placard **1622** and object **1608** are within distance **1660** of about 8 feet of each other so that the images projected onto placard **1622**

and onto the object are within field of view **1624** of projector **1506**.

[0131] Field of view **1604** of camera **1504** includes field of view **1646** and field of view **1648**. Field of view **1646** is the field of view for trigger area **1620** on the surface of object **1608** and is about 4 degrees. Field of view **1648** is the field of view for trigger area **1626** on the surface of object **1608** and is about 3 degrees. Field of view **1604** of camera **1504** is different from field of view **1624** of projector **1506** so that camera **1504** may monitor trigger areas that are outside of field of view **1624** of projector **1506**.

[0132] Referring to Figure 16B, camera **1504** is positioned generally over object **1608**. The position of camera **1504** on the *X* axis and *Z* axis is dictated by the desired positions of trigger areas. In one embodiment, camera **1504** is positioned within a range of 5 to 16 feet along the *Y* axis above object **1608** and within a range of 10 to 20 feet of floor **1610**. Trigger areas shown within the field of view of the camera. Trigger area **1620** is positioned on object **1608**. Trigger area **1626** is positioned on the surface of object **1608**. The system may project an indicator of trigger area **1626** on the surface of object **1608**.

[0133] The position of projector **1506** on the *X* and *Z* axes is dictated by the desired positions of content areas **1632** and **1634**. In one embodiment, projector **1506** is positioned within a range of 4 to 40 feet along the *Y* axis above any surface in which content is desired to be displayed. The ideal distance being dictated by the lens. The surface on which the system displays content should be within the field of view of the projector. In one embodiment, content area **1632** is positioned on the hood portion of object **1608**, which in this case is a car. Content area **1634** is positioned on placard **1622**.

[0134] Flat surface **1654**, shown in Figure 16C, is a hypothetical surface to be targeted at object **1608**. Object **1608** includes curved surface **1656**, onto which content area

1632 is projected. Angle **1658** is the difference between flat surface **1654** and curved surface **1656**.

[0135] The trigger areas are distinct from the content areas but may be linked with each other. In an alternative embodiment where a trigger area is on the headlight of a car, the associated content area may be on the hood of the car so that when the trigger area on the headlight is touched, a video is played on the content area on the hood and the audio associated with the video is played via one or more speakers. Furthermore, instead of a video being played on the hood, the trigger area on the headlight may be linked to an audio recording so that when the trigger area on the headlight is touched, the audio recording (also known as an audio response) is played.

[0136] In additional embodiments, a computer monitoring the trigger areas is also in communication with the car and can control the functions of the car in response to interaction with the trigger areas. This allows for control of the object—the car—in concert with media displayed or projected onto the object. In one embodiment, interaction with a trigger area on the hood of a car causes a video of the engine to be played on the hood and causes the system to start up the engine of the car. When the object is a car, additional features can be controlled in response to interaction with trigger areas, such as: turning on or off headlights and taillights, locking and unlocking doors, opening and closing windows, opening the trunk, and the like.

[0137] Referring to Figure 16C, surface **1654** is a hypothetical flat surface above object **1608** through which an image from the projected passes. The plane of surface **1654** is perpendicular to the direction that the projector is pointing and is tangential to object **1608**.

[0138] Referring to Figure 16D, surface **1656** is the surface of object **1608**, which is curved and not flat. Images projected onto curved surfaces, such as surface **1656**, instead of onto

flat surfaces, such as hypothetical surface **1654**, are adjusted by warping and transforming the image to match the curvature of the surface.

[0139] Referring to Figure 17A, trigger area is described. Trigger areas **1702** and **1704** are predefined 3-dimensional spaces stored in the database. Trigger areas are monitored by the infrared sensors in camera **1504** for interaction with the system by a user. When the infrared sensors detect a user in a trigger area, a predefined action, such as displaying stored content, is carried out by the system.

[0140] A trigger area can be any three-dimensional shape bounded by a series of points which form lines that enclose a volume. Referring to coordinate system **1706**, trigger area **1702** is bounded by a set of points including X_1, Y_1, Z_1 ; X_2, Y_1, Z_1 ; X_1, Y_2, Z_1 ; X_2, Y_2, Z_1 ; and function **1708**. Function **1708**, in this example, is a hyperbolic function defined by the equation

$$C \approx x^2 + y^2 \quad \text{Eq. 12}$$

[0141] Other hyperbolic functions or linear functions may be used to define any set of points, in the X , Y , or Z directions, defining a general shape, so long as the function is closed with respect to the discrete area. Trigger area **1704** is a three-dimensional polygon. The various X , Y , and Z values for each trigger area can vary. The X and Y values range between 2 and 12 inches, preferably around 7 to 8 inches. The Z value can range between 1 and 4 inches, preferably about 2 inches.

[0142] Figure 18A is a diagram of a set of data elements stored in the computer and accessed by a trigger area software program, such as touchable area application **1802**, in accordance with one embodiment. Touchable area application **1802** is an application running on a computer system that allows for the display of projected content onto objects and surfaces based user interaction with trigger areas that have been associated with and linked to the

projected content. Touchable area application **1802** includes tabs **1804**, **1806**, and **1808**; one or more content controls **1852**; and one or more trigger controls **1854**.

[0143] Tabs **1804**, **1806**, and **1808** are user interface elements displayed in the application window of touchable area application **1802** that allow for the selection between different modes of the application, such as modes **1910**, **1912**, and **1914** shown in Figure 19A. Tabs **1804**, **1806**, and **1808** are displayed in display area **1816**.

[0144] Display area **1816** is the display space of touchable area application **1802**. Touchable Area Application includes trigger controls **1854** and content controls **1852** that are displayed in display area **1816**.

[0145] Trigger controls **1854** are each associated with an instance of trigger area data structure **1818**. Trigger area data structure **1818** include several fields of data to identify a trigger area that is monitored by a camera with the computer, such as camera **1504** and computer **1502** of Figure 15.

[0146] Content controls **1852** are each associated with an instance of content data structure **1820**. Content data structure **1820** identifies the content and the content area in which the content is displayed by the system. Content data structure **1820** includes several data fields to identify the content and control its display.

[0147] Figure 18B is a diagram of a trigger area data structure in accordance with one embodiment of the disclosure. Trigger area data structure **1818** is associated with trigger control **1854** and includes coordinates **1822**, type **1824**, name **1826**, identifier **1828**, depth data **1830**, content data structure **1820**, and filename **1834**. The data associated with trigger area data structure **1818** is stored to one or more files that allows trigger data structure area **1818** to be saved and reloaded by touchable area application **1802**.

[0148] Coordinates **1822** identify x and y locations for each of four corners for where trigger control **1852** is displayed in display area **1816**.

[0149] Type **1824** identifies a type of content associated directly with trigger area data structure **1818**, such as an image or video.

[0150] Name **1826** is a user friendly name that identifies an instance of a trigger control.

[0151] Identifier **1828** is a unique identifier, such as a globally unique identifier (GUID), which is given to each trigger control **1854** of touchable area application **1802** to access trigger control **1854**.

[0152] Depth data **1830** identifies the three-dimensional location or touch box associated with trigger area data structure **1818** and trigger control **1854**.

[0153] Content data structure **1820** identifies the content area that is associated with trigger area data structure **1818** and trigger control **1854**.

[0154] Filename **1834** identifies a file, such as an image file, that provides a default image for trigger control **1854**. Filename **1834** may point to a fully transparent image so that, by default, no image is displayed by trigger control **1854**.

[0155] Figure 18C is a diagram of a content data structure in accordance with one preferred embodiment. Content data structure **1820** is associated with content control **1852** and includes several data fields to identify and display content within display area **1816** of touchable area application **1802**.

[0156] Coordinates **1836** identify x and y locations for each of the corners of content control **1852**. The corners identified by coordinates **1836** form a polygon. When the image or video associated with content data structure **1820** is displayed, the corners of the image or video

are warped and transformed to fit the shape identified by coordinates **1836**.

[0157] Type **1838** identifies the type of content associated with content data structure **1820** and content control **1852**. Content types include image, video, sound, and the like.

[0158] Loop **1840** identifies how many times to loop the content when, for example, type **1838** of content area **1820** identifies that the content is a video.

[0159] Orientation **1842** identifies the orientation used to display images associated with content control **1852**. In one embodiment, Orientation **1842** specifies that, when displayed, the image or video associated with content control **1852** is flipped about the horizontal or vertical axes or rotated by a multiple of 90 degrees.

[0160] Filename **1844** identifies the filename of the image or video associated with content data structure **1820** and content control **1852**. In additional embodiments, filename **1844** is a link to static or dynamic content such as a video file, a streaming video file, or a live camera stream that is played in response to interaction with a trigger area.

[0161] Identifier **1846** is a unique identifier or GUID that is given to each content control **1852** of touchable area application **1802** to access content control **1852**.

[0162] Name **1848** is a user friendly name that identifies an instance of a content control.

[0163] Figure 19A is a data flow diagram of a system calibrating trigger areas and content areas in accordance with the disclosure. In this embodiment, the system includes computer **1902**, camera **1904**, projector **1906**, and display **1908**.

[0164] Computer **1902** is connected to range camera **1904** via a USB (universal serial bus) 3.0 connection and is connected to projector **1906** and display **1908** by an HDMI (high-definition multimedia interface) connection. Computer **1902** controls the images displayed by

display **1908** and projector **1906**. Camera **1904** provides color data and depth data. Projector **1906** projects images sent from computer **1902** onto any surface in front of projector **1906**, including one or more display stands, objects, and screens. Display **1908** displays the application to aid the calibration process.

[0165] The touchable area application uses three modes to perform the calibration process, first mode **1910**, second mode **1912**, and third mode **1914**. When first mode **1910** is selected, user interface controls are associated with the touchable area application. For example, in the case where a static image is displayed by default and two videos are shown based on interaction with two trigger areas, the static image, the two videos, and the two trigger controls are added to the touchable area application. In First mode **1910**, the application performs steps **1916**, **1918**, **1920**, and **1922**.

[0166] In second mode **1912**, the three-dimensional real world locations for trigger areas are selected. In second mode **1912**, the application performs steps **1924**, **1926**, **1928**, **1930**, and **1932**.

[0167] In third mode **1914**, the projected locations of the projected content is selected, fine-tuned, and aligned. In third mode **1914** the application performs steps **1934**, **1936**, **1938**, and **1940**.

[0168] At step **1916**, a first tab is selected in the touchable area application to set the mode of the touchable area application to first mode **1910**. One example of code for selecting a first tab is shown in the computer program listing appendix in the computer program listing of the file named MainWindow.xaml.cs at lines 922-926.

[0169] At step **1918**, the display image of the application is sent to display **1908** so that the user interface controls—trigger controls and content controls—can be added and

manipulated.

[0170] At step **1920**, trigger controls and content controls are added and updated. New trigger controls and content controls may be added and the associations between the trigger controls, trigger areas, content controls, and content areas may be updated. The corners of the content controls and trigger controls can be moved or dragged to reshape the images and video that are projected onto the content areas and trigger areas. One example of code that performs this function is shown in the computer program listing appendix in the computer program listing of the file named NonAffineTransformControl.xaml.cs at lines 437-476. This code calls a function to determine a transformation matrix, which is shown in the computer program listing appendix in the computer program listing of the file named NonAffineTransformControl.xaml.cs at lines 1220-1258. One example of code for updating trigger or content controls is shown in the computer program listing appendix in the computer program listing of the file named NonAffineTransformControl.xaml.cs at lines 478-512.

[0171] At step **1922**, for each change to the user interface controls from step **1920**, the image sent to display **1908** is updated. This provides feedback to allow the sequences, timing, and interaction between the trigger controls and content controls.

[0172] At step **1924**, a second tab is selected to shift the application to second mode **1912** to allow for the selection of the physical locations of the trigger areas that are associated with trigger controls. One example of code for selecting a second tab is shown in the computer program listing appendix in the computer program listing of the file named MainWindow.xaml.cs at lines 927-971.

[0173] At step **1926**, computer **1902** receives color and depth data from camera **1904**.

[0174] At step **1928**, the display area of the application window of the application is

changed to show a combination of the color and depth data received from camera **1904** with the trigger controls overlaid onto the image.

[0175] At step **1930**, the trigger controls that were added in first mode **1910** are associated physical locations that are monitored by camera **1904**. The associations are created by dragging and moving the trigger controls displayed on display **1908**. One example of code for updating trigger controls is shown in the computer program listing appendix in the computer program listing of the file named NonAffineTransformControl.xaml.cs at lines 478-512.

[0176] At step **1932**, for each change to a trigger control and each change in the color and depth data received from camera **1904**, computer **1902** updates the image sent to display **1908**.

[0177] At step **1934**, a third tab, such as third tab **1808** of Figure 18, is selected to set the mode of the application to third mode **1914** to allow for selection of the location of the content areas that will be associated with the content controls. One example of code for selecting a third tab is shown in the computer program listing appendix in the computer program listing of the file named MainWindow.xaml.cs at lines 916-920.

[0178] At step **1936**, computer **1902** stops displaying the color and depth data from camera **1904** and displays the touchable area application as a full screen window on projector **1906** including the user interface controls. Projector **1906** projects the image from computer **1902** onto the surfaces and objects in front of projector **1906**. For example, when camera **1904** projects onto the hood of a car the content displayed by the touchable area application is an image of the engine. In an alternative embodiment, the image sent by computer **1902** to projector **1906** is also sent to display **1908** so that the image is displayed by both display **1908** and by projector **1906**.

[0179] At step **1938**, the content control is updated. For example, the sizes of the content control are adjusted to match the surfaces onto which they are being projected. When the hood of the car has a trapezoidal shape, the corners of the content control are dragged to match the trapezoidal shape of the hood. One example of code for updating content controls is shown in the computer program listing appendix in the computer program listing of the file named NonAffineTransformControl.xaml.cs at lines 478-512.

[0180] At step **1940**, for each update to the content controls from step **1938**, the image sent to the projector is updated to include the changes made in step **1938**.

[0181] Figure 19B is a flow chart for updating a trigger area associated with a trigger control in accordance with one embodiment of the disclosure. The method for updating a trigger area of Figure 19B is one embodiment for step **1930** of Figure 19A.

[0182] At step **1942**, color and depth data are received from a camera, such as camera **1904**. The color and depth data are received as one or more byte streams that contain two frames of data, one based on the color sensor of the camera and one based on the infrared depth sensor of the camera. The resolution of the color sensor and the depth sensor differ so that the resolution of the two frames of data also differ. As one example, the color data may have a resolution of 1920 by 1080 with 32 bits of color information per pixel and the depth data may have a resolution of 640 by 480 with 16 bits of depth information per pixel.

[0183] At step **1944**, a combination of the color and depth data is displayed. The color data resolution is cropped to the depth data resolution. Within the cropped resolution, color data is displayed if the depth data indicates there is a surface or object within the range of the depth sensor.

[0184] At step **1946**, a trigger area associated with a trigger control is selected. The

trigger area identifies the three-dimensional coordinates of a trigger area that is stored in a trigger area data structure associated with a trigger control and is with respect to the camera that provides the depth data. As shown in Figure 19C, area **1952** is an area of a surface within the range and viewing area of the depth sensor. In a preferred embodiment, the shape of the area defaults to a square or rectangle, but any other shape can be used.

[0185] At step **1948**, a depth for the trigger area is selected. The depth combined with the trigger area creates a volume that is associated with a trigger area and is used to determine if the trigger area has been touched. Figure 19D shows volume **1954** created from area **1952** through the addition of depth **1956**.

[0186] At step **1950**, the trigger area and depth are stored. The trigger area and depth are associated with the trigger area and saved to the database. In a preferred embodiment, the trigger area and depth are stored in depth data **1830** of trigger area data structure **1818** of Figure 18B.

[0187] Figure 19E is a flow chart for updating a content area in accordance with one embodiment of the disclosure. The method for updating a content area of Figure 19E is one embodiment for step **1938** of Figure 19A.

[0188] At step **1958**, an initial flat projection area is selected. The native resolution of the content that is selected and associated with a content control provides the initial flat projection area. Figure 19F shows initial flat projection **1968** of content that has been selected.

[0189] At step **1960**, an area related to an affine transformation is selected. In a preferred embodiment, the area for the affine transformation is selected by dragging the corners the content area into a desired shape. As an example, when the content area is a rectangular image, one or more of the corners are dragged to form a scaled parallelogram shape, as shown by

transformation **1970** of Figure 19F.

[0190] At step **1962**, an area related to a non-affine transformation is selected. In a preferred embodiment, the area is selected by dragging one or more of the corners and lines that from the boundary of the content area into curved shapes, as shown by transformation **1972** of Figure 19F.

[0191] At step **1964**, a brush or filter is created from the affine transformation and the non-affine transformation. The brush will be applied to the content associated with the content control to transform the original size and shape of the content to the selected size and shape of the area selected via steps **1960** and **1962** that correspond to an affine transformation and a non-affine transformation. When the content is a video, the brush is applied to each frame of the video as the video is displayed.

[0192] At step **1966**, the brush created from the selected areas for the affine transformation and the non-affine transformation are saved to persistent storage

[0193] Figure 19F shows the matrix transformations applied to content in accordance with one embodiment of the disclosure. Flat projection **1968** is an initial projection that has not been warped and transformed. Projection **1970** has been warped and transformed using an affine transformation. Projection **1972** has been warped and transformed using a non-affine transformation.

[0194] The following code listing provides an example of calculating a transformation:

```
// The input array of points describes a 2D rectangle
// (with Z assumed to be zero) in the order
// lower-left, upper-left, lower-right, upper-right.
```

```

// The returned transform maps the points (0, 0, 0),
// (0, 1, 0), (1, 0, 0), and (1, 1, 0) to these points.

private Matrix3D CalculateNonAffineTransform(Point3D[] points)
{
    // Affine transform
    // -----

    // This matrix maps (0, 0) --> (x0, y0)
    //          (0, 1) --> (x1, y1)
    //          (1, 0) --> (x2, y2)
    //          (1, 1) --> (x2 + x1 + x0, y2 + y1 + y0)

    Matrix3D A = new Matrix3D();

    A.M11 = points[2].X - points[0].X;
    A.M12 = points[2].Y - points[0].Y;
    A.M21 = points[1].X - points[0].X;
    A.M22 = points[1].Y - points[0].Y;
    A.OffsetX = points[0].X;
    A.OffsetY = points[0].Y;

    // Calculate point (a, b) that get mapped by the affine
    // transform to (x3, y3)

    double den = A.M11 * A.M22 - A.M12 * A.M21;

    double a = (A.M22 * points[3].X - A.M21 * points[3].Y +
                A.M21 * A.OffsetY - A.M22 * A.OffsetX) / den;

```

```

double b = (A.M11 * points[3].Y - A.M12 * points[3].X +
            A.M12 * A.OffsetX - A.M11 * A.OffsetY) / den;

// Non-affine transform

// -----

// This matrix maps (0, 0) --> (0, 0)

//      (0, 1) --> (0, 1)

//      (1, 0) --> (1, 0)

//      (1, 1) --> (a, b)

Matrix3D B = new Matrix3D();

B.M11 = a / (a + b - 1);

B.M22 = b / (a + b - 1);

B.M14 = B.M11 - 1;

B.M24 = B.M22 - 1;

return B * A;

}

```

[0195] Figure 20A is a data flow diagram of a system using trigger areas in accordance with the disclosure. The system includes computer **2002**, range camera **2004**, and projector **2006** and has already been calibrated to align the trigger areas and content areas to the surfaces and objects in front of projector **2006**.

[0196] Computer **2002** is connected to range camera **2004** via a USB (universal serial bus) 3.0 connection and is connected to projector **2006** by an HDMI (high-definition multimedia interface) connection. Computer **2002** controls the image displayed by projector **2006** based on data received from range camera **2004**. Range camera **2004** provides color data and depth data. Projector **2006** projects images sent from computer **2002** onto any surface in front of projector **2006**, including one or more display stands, objects, and screens.

[0197] At step **2008**, computer **2002** sends the display image that will be received and displayed by projector **2006**. The display image has a resolution of, e.g., 1920 by 1080 pixels, which is a native resolution for projector **2006**.

[0198] At step **2010**, a frame of data is sent from range camera **2004** and received by computer **2002**. The color data and the depth data are measured by range camera **2004** and are separated into two different byte streams with different resolutions. The color data is 1920 by 1080 by 32 bits with 8 bits for each of red, green, blue, and alpha channels. The alpha channel defaults to 1. The depth data is 640 by 480 by 16 bits with 13 bits for depth information that measures depth from range camera **2004** in 1 millimeter increments.

[0199] At step **2012**, computer **2002** determines if a trigger area has been touched. The determination is made by comparing a base line of the depth data received from range camera **2004** with the current depth data received from range camera **2004**. In one preferred embodiment, the baseline depth data is an average of three frames of data captured during the calibration process.

[0200] In one embodiment, detection of a trigger area is determined by calculating whether 20% or more of a volume associated with the trigger area is occupied for at least three frames in a row. One example of code for performing this function is shown in the computer

program listing appendix in the computer program listing of the file named MainWindow.xaml.cs at lines 494-527. The occupation of the volume is determined by the depth associated with the pixels related to the volume of the trigger area. The depth should be less than the baseline depth, but greater than a threshold depth so that a trigger area is only activated by an object that is near the real world location of the trigger area.

[0201] In another embodiment, for each pixel of a trigger area that is occupied, the surrounding eight pixels (forming a 3 by 3 or nine pixel matrix) are tested to see if they are each similarly occupied. This additional precision allows for the touch of a single finger to activate a trigger area.

[0202] At step **2014**, computer **2002** updates the display image to be sent to projector **2006** based on frame data received from range camera **2004**. When no trigger areas are clicked or activated, the display image is updated for content areas that includes video. When a trigger area is properly activated, the content areas associated with that trigger area are activated.

[0203] At step **2016**, computer **2002** sends the updated image. The updated image is based on the content being displayed and the interaction with trigger areas.

[0204] Figure 20B is a flow chart for determining a touch of a trigger area in accordance with one embodiment of the disclosure. One example of code for performing steps analogous to the steps of the flow chart of Figure 20B is shown in the computer program listing appendix in the computer program listing of the file named MainWindow.xaml.cs at lines 494-527.

[0205] At step **2018**, depth data is received from a camera, such as range camera **2004** of Figure 20. In a preferred embodiment, the depth data has a resolution of 640 by 480 pixels with 13 bits of depth information per pixel and aligned to 16 bits.

[0206] At step **2020**, a determination is made as to whether 20% or more of the trigger volume associated with a trigger area has been filled by an object. Each pixel of the depth data related to the trigger volume is compared to its corresponding baseline depth to determine if the pixel includes an object that is a threshold amount above the baseline depth. When 20% or more of the pixels associated with the trigger volume have been determined to include an object, then a frame counter is incremented and the method proceeds to step **2022**. Otherwise, the frame counter is decremented to a minimum value of zero and the method returns to step **2018** to wait for the next frame of depth data.

[0207] At step **2022**, the frame count for the trigger area is compared. In a preferred embodiment, an object must be detected in relation to the touchable area for three consecutive frames of data. When it has been detected for three consecutive frames of data, the method proceeds to step **2024**. Otherwise, the method returns to step **2018** to wait for the next frame of depth data.

[0208] At step **2024**, an object of sufficient size has been detected a sufficient number of times to activate a touch event that is associated with the trigger area. In a preferred embodiment the touch event is a mouse click event that is associated with a trigger control associated with the trigger area that causes the display of content via a content control onto the content area. The content displayed in the content area may be static, dynamic, or streamed.

[0209] In Figures 21A and 21B, a system is being calibrated to display content onto an object. A touchable area application, such as touchable area application **1802** of Figure 18, includes tabs **2116**, **2118**, and **2120** that allow for switching between different modes—such as modes **1910**, **1912**, and **1914** of Figure 19A—in the touchable area application for calibration. The first mode to add, associate, and manipulate trigger areas and content areas is accessed by

selecting tab **2116**. The second mode to identify the three-dimensional real world location of the trigger areas associated with trigger controls is accessed by selecting tab **2118**. The third mode to display, align, and manipulate the content areas with respect to the object is accessed by selecting tab **2120**.

[0210] Figure 21A is a view of the application when a first tab is selected. When first tab **2116** is selected, the application window **2102** is seen on a first display, which is connected to and may be substantially co-located with a computer, such as computer **1902** of Figure 19A. Displaying application window **2102** on the first display allows a user to set up the trigger areas and content areas for the system along with the linking and interaction behaviors associated with the trigger controls, content controls, trigger areas, and content areas. Trigger controls **2104** and **2106** are displayed in application window **2102**. Application window **2102** also displays content control **2108**, content control **2110**, content control **2112** and mouse pointer **2114**. Application window **2102** also includes tabs **2116**, **2118**, and **2120**. Background **2128** of application window **2102** is black.

[0211] Trigger control **2104** is associated with content control **2110**. When the system is running and trigger control **2104** is touched, the content of content control **2110** will be played on top of the content of content control **2108**. Similarly, trigger control **2106** is associated with content control **2112** so that touching or clicking trigger control **2106** plays the media or content of content control **2112** on top of the content of content control **2108**.

[0212] Content control **2108** includes the default content that is normally displayed by the system. In the embodiment of Figures 21A and 21B, content control **2108** is linked to a static image of an engine, such as the engine of car **2122**. Content control **2110** is linked to trigger control **2104** and includes a video of the performance characteristics of the object, i.e., car

2122 of Figures 21B. Content control **2112** is linked to a video of the safety systems of car **2122** and is associated with trigger control **2106**.

[0213] Figure 21B is a view of the application when second tab **2118** is selected. When second tab **2118** is selected, instead of showing content controls on background **2128**, application window **2102** shows a combination of color and depth data from the camera. When there is no depth data for a pixel, the pixel is shown as black. Otherwise, the RGB (red green blue) color value from the camera is shown. In one embodiment when the resolution of the color data does not match the resolution of the depth data, parts of the image where there is no color data are shown only using the depth data, such as in areas **2130** and **2132**. Area **2134** is shown using the combination of color and depth data and areas **2130** and **2132** are shown using only depth data since there is no color data for that portion. Alternative embodiments show any one or combination of color data and depth data on application window **2102**.

[0214] Trigger controls **2104** and **2106** can be dragged to any part of the view shown application window **2102** in Figure 21B. When a trigger control is moved, it is associated with the three-dimensional physical location that corresponds with the pixels displayed in application windows **2102** in Figure 21B. A touch box is created for the trigger area associated with the trigger control so that when it is determined that an object sufficiently fills the touch box, the trigger control is activated by creating a virtual mouse click for the trigger control associated with the trigger area in the display area of the touchable area application.

[0215] In an additional embodiment, when the projector can display an image in view of the camera, the projector flashes an image related to the size and location of the trigger area. When the flashing image related to the trigger area is identified from the color data from the camera, the three dimensional coordinates that correspond to the trigger area are identified from

the depth data and are associated with the trigger control to automatically calibrate the three dimensional location of the trigger area with the trigger control.

[0216] It will be appreciated by those skilled in the art that changes could be made to the embodiments described above without departing from the broad inventive concept. It is understood, therefore, that this disclosure is not limited to the particular embodiments herein, but it is intended to cover modifications within the spirit and scope of the present disclosure as defined by the appended claims.

```

1.    using System;
2.    using System.Collections.Generic;
3.    using System.Configuration;
4.    using System.IO;
5.    using System.Linq;
6.    using System.Runtime.InteropServices;
7.    using System.Text;
8.    using System.Threading;
9.    using System.Threading.Tasks;
10.   using System.Windows;
11.   using System.Windows.Controls;
12.   using System.Windows.Data;
13.   using System.Windows.Documents;
14.   using System.Windows.Input;
15.   using System.Windows.Media;
16.   using System.Windows.Media.Imaging;
17.   using System.Windows.Media.Media3D;
18.   using System.Windows.Navigation;
19.   using System.Windows.Shapes;
20.   using System.Windows.Threading;
21.   using System.Xml;
22.   using System.Xml.Serialization;
23.   using WpfAnimatedGif;
24.
25.
26.   using Microsoft.Kinect;
27.   using System.Runtime.InteropServices;
28.   using System.Diagnostics;
29.
30.   namespace NonAffineTransform
31.   {
32.
33.       public class WispInteractiveData
34.       {
35.           public string wispID { get; set; }
36.           public int wispIndex { get; set; }
37.           public int wispPixDepth { get; set; }
38.
39.       }
40.       public class WispCoveredFrameItem
41.       {
42.           //wispID we are working with
43.           public string wispID { get; set; }
44.           //Captures all the pixels we are worrying about (helps in
avg of how many are covered
45.           public int totalPixels { get; set; }
46.           //for the loop = to count how many of the pixels for this
wisp have been covered.
47.           public int currentPixCount { get; set; }
48.           //CountNum - must have at least a certain coverage for a
certain number of frames to be considered a click
49.           public int frameCountNum { get; set; }
50.           //after click is achieved, then this is set to true. a new
click cannot happen while this is true

```

```

51.         //set to false after countNum hits 0 so we can have a new
        click.
52.         public bool isCurrentlyInClickState { get; set; }
53.     }
54.
55.
56.
57.     public partial class MainWindow : Window
58.     {
59.         /// <summary>
60.         /// Active Kinect sensor
61.         /// </summary>
62.         private KinectSensor kinectSensor = null;
63.
64.         private const int MapDepthToByte = 8000 / 256;
65.
66.         /// <summary>
67.         /// Reader for depth frames
68.         /// </summary>
69.         ///
70.         private CoordinateMapper coordinateMapper = null;
71.
72.         private MultiSourceFrameReader multiFrameSourceReader =
        null;
73.
74.         /// <summary>
75.         /// Description of the data contained in the depth frame
76.         /// </summary>
77.         private FrameDescription depthFrameDescription = null;
78.
79.         /// <summary>
80.         /// Bitmap to display
81.         /// </summary>
82.         private WriteableBitmap depthBitmap = null;
83.
84.         /// <summary>
85.         /// Intermediate storage for frame data converted to color
86.         /// </summary>
87.         public static byte[] depthPixels = null;
88.
89.         //displayMe
90.         ushort[] _depthData = null;
91.         byte[] _colorData = null;
92.         byte[] _displayPixels = null;
93.         ColorSpacePoint[] _colorPoints = null;
94.         readonly int BYTES_PER_PIXEL =
        (PixelFormat.Bgr32.BitsPerPixel + 7) / 8;
95.         WriteableBitmap _bitmap = null;
96.
97.         int depthWidth = 0;
98.         int depthHeight = 0;
99.         int colorWidth = 0;
100.        int colorHeight = 0;
101.

```

```

102.         //inverse arrays
103.         byte[] InverseDepthPixels = null;
104.
105.
106.         /// <summary>
107.         /// Current status text to display
108.         #region "initSpaceeVars"
109.
110.         //READ in default settings -these can be changed in the
         app.config file
111.         private int FRAME_NUM_REQUIRED_FOR_CLICK =
         int.Parse(ConfigurationManager.AppSettings["conf_FRAME_NUM_REQUIRED_
         FOR_CLICK"]);
112.         private double WISP_PERCENT_COVERAGE_FOR_CLICK =
         double.Parse(ConfigurationManager.AppSettings["conf_WISP_PERCENT_COV
         ERAGE_FOR_CLICK"]); // percentage
113.         public int MIN_DEPTH_CLICK =
         int.Parse(ConfigurationManager.AppSettings["conf_MIN_DEPTH_CLICK"]);
114.         public int MAX_DEPTH_CLICK =
         int.Parse(ConfigurationManager.AppSettings["conf_MAX_DEPTH_CLICK"]);
115.         public int FRAME_SKIP =
         int.Parse(ConfigurationManager.AppSettings["conf_FRAME_SKIP"]);
116.         public int FRAME_SKIP_COUNTER = 0;
117.
118.         List<WispCoveredFrameItem> WispViewTabList = new
         List<WispCoveredFrameItem>();
119.
120.         public bool PauseTouch = false;
121.         public int PauseTouchCounter = 0;
122.         public int PauseOnLongWaitCounter = 0;
123.         public int PauseOnLongWaitMax = 1500;
124.
125.         public static bool enterTab2 = false;
126.         public static bool enterViewerMode = false;
127.
128.         public static string popupRecieve_WispName = "";
129.         public static List<Point> recievePop_tempPoints = new
         List<Point>();
130.         public static bool processPopup = false;
131.
132.         private List<WispInteractiveData> wispDepths = new
         List<WispInteractiveData>();
133.         public static int theBitmapSourceWidth;
134.
135.         //public bool startInViewMode = false;
136.         private string _WispCoveredFrameItem_XMLFile =
         "WispCoveredFrameItem.xml";
137.         private string _WispDepths_XMLFile = "WispDepths.xml";
138.
139.         #endregion
140.
141.         public MainWindow()
142.         {
143.             AppDomain currentDomain = AppDomain.CurrentDomain;

```

```

144.         currentDomain.UnhandledException += new
        UnhandledExceptionHandler(KillProgram);
145.
146.         //start 3d Camera
147.         startKinect();
148.         InitializeComponent();
149.         //load in all possible touch points from the xml file
150.         LoadWispDepthsFromXML();
151.         LoadWispsFromXML();
152.
153.         //check if monoter system issues this program a command
154.         var sivm = App.startInViewMode;
155.
156.         if (sivm == true)
157.         {
158.             //load viewer and full screen.
159.             //System.Windows.MessageBox.Show("true");
160.             startAppinFullView();
161.         }
162.     }
163.
164.     static void KillProgram(object sender,
        UnhandledExceptionEventArgs args)
165.     {
166.
167.         Environment.Exit(0);
168.         //Exception e = (Exception)args.ExceptionObject;
169.         //Console.WriteLine("MyHandler caught : " + e.Message);
170.         //Console.WriteLine("Runtime terminating: {0}",
        args.IsTerminating);
171.     }
172.
173.     public void startKinect()
174.     {
175.         // get the kinectSensor object
176.         this.kinectSensor = KinectSensor.GetDefault();
177.
178.
179.         // open the reader for the depth frames
180.
181.         this.multiFrameSourceReader =
        this.kinectSensor.OpenMultiSourceFrameReader(FrameSourceTypes.Depth
        | FrameSourceTypes.Color);
182.
183.
184.         this.multiFrameSourceReader.MultiSourceFrameArrived +=
        this.Reader_MultiSourceFrameArrived;
185.
186.         this.coordinateMapper =
        this.kinectSensor.CoordinateMapper;
187.
188.         // get FrameDescription from DepthFrameSource
189.         this.depthFrameDescription =
        this.kinectSensor.DepthFrameSource.FrameDescription;

```

```

190.         depthWidth = depthFrameDescription.Width;
191.         depthHeight = depthFrameDescription.Height;
192.
193.         // allocate space to put the pixels being received and
converted
194.         depthPixels = new byte[this.depthFrameDescription.Width
* this.depthFrameDescription.Height];
195.
196.         FrameDescription colorFrameDescription =
this.kinectSensor.ColorFrameSource.FrameDescription;
197.
198.
199.         colorWidth = colorFrameDescription.Width;
200.         colorHeight = colorFrameDescription.Height;
201.
202.
203.         //displayMe
204.         if (_displayPixels == null)
205.         {
206.             _depthData = new ushort[depthWidth * depthHeight];
207.             _colorData = new byte[colorWidth * colorHeight *
BYTES_PER_PIXEL];
208.             _displayPixels = new byte[depthWidth * depthHeight *
BYTES_PER_PIXEL];
209.             _colorPoints = new ColorSpacePoint[depthWidth *
depthHeight];
210.         }
211.         _bitmap = new WriteableBitmap(depthWidth, depthHeight,
96.0, 96.0, PixelFormats.Bgra32, null);
212.
213.
214.         //inverses
215.         InverseDepthPixels = new byte[depthWidth * depthHeight];
216.
217.
218.         // create the bitmap to display
219.         this.depthBitmap = new
WriteableBitmap(this.depthFrameDescription.Width,
this.depthFrameDescription.Height, 96.0, 96.0, PixelFormats.Gray8,
null);
220.
221.
222.         // open the sensor
223.         this.kinectSensor.Open();
224.     }
225.
226.     //for each frame I can read - do this
227.     private void Reader_MultiSourceFrameArrived(object sender,
MultiSourceFrameArrivedEventArgs e)
228.     {
229.
230.
231.
232.

```

```

233.         bool depthFrameProcessed = false;
234.         DepthFrame df = null;
235.         ColorFrame cf = null;
236.         bool isBitmapLocked = false;
237.
238.         MultiSourceFrame multiSourceFrame =
e.FrameReference.AcquireFrame();
239.         if (PauseTouch == true)
240.         {
241.             PauseTouchCounter++;
242.             if (PauseTouchCounter > 60)
243.                 PauseTouch = false;
244.
245.         }
246.         else
247.         {
248.             PauseOnLongWaitCounter++;
249.             if (PauseOnLongWaitCounter > PauseOnLongWaitMax)
250.             {
251.                 PauseTouch = true;
252.                 PauseOnLongWaitCounter = 0;
253.             }
254.
255.             // txtDebug.Text = "Entered Reader";
256.
257.             // If the Frame has expired by the time we process
this event, return.
258.             if (multiSourceFrame == null)
259.             {
260.                 return;
261.             }
262.
263.             // We use a try/finally to ensure that we clean up
before we exit the function.
264.             // This includes calling Dispose on any Frame
objects that we may have and unlocking the bitmap back buffer.
265.             try
266.             {
267.                 df =
multiSourceFrame.DepthFrameReference.AcquireFrame();
268.                 cf =
multiSourceFrame.ColorFrameReference.AcquireFrame();
269.
270.                 if (FRAME_SKIP_COUNTER > FRAME_SKIP)
271.                 {
272.                     FRAME_SKIP_COUNTER = 0;
273.
274.
275.                     // // If any frame has expired by the time we
process this event, return.
276.                     // // The "finally" statement will Dispose
any that are not null.
277.                     if ((df == null) || (cf == null))
278.                     {

```

```

279.         return;
280.     }
281.
282.     if (enterTab2 == true)
283.     {
284.         //Inverse Streams
285.         // depth stream
286.
287.
288.         //displayMe
289.         if (((depthWidth * depthHeight) ==
_depthData.Length) && ((colorWidth * colorHeight * BYTES_PER_PIXEL)
== _colorData.Length))
290.         {
291.             df.CopyFrameDataToArray(_depthData);
292.
293.             if (cf.RawColorImageFormat ==
ColorImageFormat.Bgra)
294.             {
295.                 cf.CopyRawFrameDataToArray(_colorData);
296.             }
297.             else
298.             {
299.                 cf.CopyConvertedFrameDataToArray(_colorData, ColorImageFormat.Bgra);
300.             }
301.
302.
303.
304.             this.coordinateMapper.MapDepthFrameToColorSpace(_depthData,
_colorPoints);
305.
306.             Array.Clear(_displayPixels, 0,
_displayPixels.Length);
307.
308.             for (int y = 0; y < depthHeight; ++y)
309.             {
310.                 for (int x = 0; x < depthWidth; ++x)
311.                 {
312.                     int depthIndex = (y *
depthWidth) + x;
313.
314.
315.                     ColorSpacePoint colorPoint =
_colorPoints[depthIndex];
316.
317.                     int colorX =
(int)Math.Floor(colorPoint.X + 0.5);
318.                     int colorY =
(int)Math.Floor(colorPoint.Y + 0.5);
319.

```

```

320.             if ((colorX >= 0) && (colorX <
colorWidth) && (colorY >= 0) && (colorY < colorHeight))
321.             {
322.                 int colorIndex = ((colorY *
colorWidth) + colorX) * BYTES_PER_PIXEL;
323.                 int displayIndex =
depthIndex * BYTES_PER_PIXEL;
324.
325.                 _displayPixels[displayIndex
+ 0] = _colorData[colorIndex];
326.                 _displayPixels[displayIndex
+ 1] = _colorData[colorIndex + 1];
327.                 _displayPixels[displayIndex
+ 2] = _colorData[colorIndex + 2];
328.                 _displayPixels[displayIndex
+ 3] = 0xff;
329.             }
330.
331.         }
332.     }
333.
334.
335.     }
336. }
337.
338.
339.
340.     using (DepthFrame depthFrame = df)
341.     {
342.         if (depthFrame != null)
343.         {
344.             // the fastest way to process the body
index data is to directly access
345.             // the underlying buffer
346.             using (Microsoft.Kinect.KinectBuffer
depthBuffer = depthFrame.LockImageBuffer())
347.             {
348.                 // verify data and write the color
data to the display bitmap
349.                 if
(((this.depthFrameDescription.Width *
this.depthFrameDescription.Height) == (depthBuffer.Size /
this.depthFrameDescription.BytesPerPixel)) &&
350.                 (this.depthFrameDescription.Width == this.depthBitmap.PixelWidth) &&
(this.depthFrameDescription.Height == this.depthBitmap.PixelHeight))
351.                 {
352.                     // Note: In order to see the
full range of depth (including the less reliable far field depth)
353.                     // we are setting maxDepth to
the extreme potential depth threshold
354.                     ushort maxDepth =
ushort.MaxValue;
355.

```

```

356.                                     // If you wish to filter by
    reliable depth distance, uncomment the following line:
357.                                     ///// maxDepth =
    depthFrame.DepthMaxReliableDistance
358.
359.
    this.ProcessDepthFrameData(depthBuffer.UnderlyingBuffer,
    depthBuffer.Size, depthFrame.DepthMinReliableDistance, maxDepth);
360.                                     depthFrameProcessed = true;
361.                                     }
362.
363.
364.
365.                                     }
366.                                     }
367.                                     }
368.
369.                                     //Inverse Depth Array to match Picture
370.                                     for (int y = 0; y < depthHeight; ++y)
371.                                     {
372.                                         int counter = depthWidth - 1;
373.                                         for (int x = 0; x < depthWidth; x++)
374.                                         {
375.                                             int depthIndex = (y * depthWidth) + x;
376.                                             int reverseIndex = (y * depthWidth) +
counter;
377.                                             InverseDepthPixels[depthIndex] =
depthPixels[reverseIndex];
378.                                             counter--;
379.                                         }
380.                                     }
381.                                     depthPixels = InverseDepthPixels;
382.                                     InverseDepthPixels = new byte[depthWidth *
depthHeight];
383.
384.
385.
386.                                     // var r = wispDepths;
387.                                     if (processPopup == true)
388.                                     {
389.
390.                                         //Remove the wisp if it exists before adding
it back. (so when you move around, it doesn't ahve 1000 copies -
just one)
391.                                         wispDepths.RemoveAll(c => c.wispID ==
popupRecieve_WispName);
392.                                         WispViewTabList.RemoveAll(c => c.wispID ==
popupRecieve_WispName);
393.
394.                                         //add all points from the clicked wisp to
the list.
395.                                         foreach (Point pt in recievePop_tempPoints)
396.                                         {
397.

```

```

398.             int pixelIndex = (int)(pt.X + ((int)pt.Y
    * this.depthBitmap.PixelWidth));
399.
400.
401.             int depth = 0;
402.             if (depthPixels[pixelIndex] > 0)
403.                 depth = depthPixels[pixelIndex];
404.             else
405.                 depth = 0;
406.
407.
408.
409.             WispInteractiveData wid = new
WispInteractiveData();
410.             wid.wispID = popupRecieve_WispName;
411.             wid.wispIndex = pixelIndex;
412.             wid.wispPixDepth = depth;
413.             wispDepths.Add(wid);
414.
415.
416.         }
417.
418.         //Remove any Dups
419.         //wispDepths =
wispDepths.Distinct().ToList();
420.
421.
422.         //setup this wisp in the wisp list
423.         WispCoveredFrameItem itd = new
WispCoveredFrameItem();
424.         itd.wispID = popupRecieve_WispName;
425.         itd.frameCountNum = 0;
426.         itd.isCurrentlyInClickState = false;
427.         itd.currentPixCount = 0;
428.         itd.totalPixels = wispDepths.Where(x =>
x.wispID == popupRecieve_WispName).Count();
429.         WispViewTabList.Add(itd);
430.
431.
432.         //clear
433.         //var t = this.bitmap.PixelWidth;
434.         var j = "Pause";
435.         popupRecieve_WispName = "";
436.         recievePop_tempPoints.Clear();
437.         processPopup = false;
438.
439.         //Build XML Values and save to XML files
440.
441.         SaveWispsToXML();
442.         SaveWispDepthsToXML();
443.
444.
445.
446.

```

```

447.         }
448.         if (enterViewerMode == true)
449.         {
450.             //use WispDepths list to loop through all
coordinates and
451.             //ex: if 25% of the depth of a wisp is
greater than 50mm - trigger a click.
452.             //you may have to add a touchcount (like
greenscreens on the spacescreens) to make sure a consistant 25% is
covered over three or four frames.
453.             // use WispCoveredFrameCountList for
external touchcounts.
454.             //finally, you might have to
455.
456.             //FRAME_NUM_REQUIRED_FOR_CLICK = 5;
457.             //WISP_PERCENT_COVERAGE_FOR_CLICK = 25; //
percentage
458.             //WispViewTabList should be filled with
names, but no data.
459.
460.             WispViewTabList.Select(c => {
c.currentPixCount = 0; return c; }).ToList();
461.
462.
463.             //string wispDepthListOutput = "";
464.
465.             foreach (var di in wispDepths)
466.             {
467.
468.
469.                 //throw out the 0's
470.                 if ((di.wispPixDepth > 0) &&
(di.wispIndex > 0))
471.                 {
472.
473.                     int depth =
depthPixels[di.wispIndex];
474.                     int BaseLineDepth = di.wispPixDepth;
475.
476.                     if (depth > 1 && BaseLineDepth > 1)
477.                     {
478.                         var p1 =
depthPixels[di.wispIndex];
479.                         if ((BaseLineDepth - depth >
MIN_DEPTH_CLICK) && (BaseLineDepth - depth < MAX_DEPTH_CLICK))
480.                         {
481.                             //count this one point as a
click.
482.                             WispViewTabList.First(w =>
w.wispID == di.wispID).currentPixCount++;
483.
484.                             // var tt = 0;
485.                         }
486.                     }

```

```

487.
488.
489.         }
490.         // wispDepthListOutput +=
di.wispIndex.ToString() + ": " + di.wispPixDepth.ToString() +
"\r\n";
491.     }
492.
493.         //all touch as been calculated - time to
render a click at the right time.
494.         foreach (var wtli in WispViewTabList)
495.         {
496.             if (wtli.currentPixCount >
(wtli.totalPixels * WISP_PERCENT_COVERAGE_FOR_CLICK))
497.             {
498.                 wtli.frameCountNum++;
499.             }
500.             else
501.             {
502.                 wtli.frameCountNum--;
503.             }
504.
505.             //click OR reset
506.             if (wtli.frameCountNum >=
FRAME_NUM_REQUIRED_FOR_CLICK)
507.             {
508.
509.                 //VIRTUAL Click
510.
uc3.VirtualWispClick(wtli.wispID.ToString());
511.
512.                 wtli.frameCountNum = 0;
513.                 wtli.currentPixCount = 0;
514.                 PauseTouchCounter = 0;
515.                 PauseTouch = true;
516.                 PauseOnLongWaitCounter = 0;
517.                 //}
518.             }
519.             else if (wtli.frameCountNum <= 0)
520.             {
521.                 //CLICK RESET
522.                 wtli.frameCountNum = 0;
523.                 wtli.isCurrentlyInClickState =
false;
524.             }
525.
526.
527.         }
528.
529.
530.
531.
532.
533.     }

```

```

534.
535.
536.         if (depthFrameProcessed && enterTab2 == true)
537.         {
538.
539.             //debug
540.             //foreach (var item in wispDepths)
541.             //{
542.                 //    depthPixels[item.wispIndex] = 1;
543.
544.                 //    //displayMe
545.                 //    int displayIndex = item.wispIndex *
BYTES_PER_PIXEL;
546.
547.                 //    _displayPixels[displayIndex + 0] =
0xff;
548.                 //    _displayPixels[displayIndex + 1] =
0xff;
549.                 //    _displayPixels[displayIndex + 2] =
0xff;
550.                 //    _displayPixels[displayIndex + 3] =
0xff;
551.                 //}
552.
553.
554.
555.
556.                 this.RenderDepthPixels();
557.             }
558.
559.
560.         } // END FRAME_SKIP
561.         else
562.         {
563.             FRAME_SKIP_COUNTER++;
564.             if (df != null)
565.             {
566.                 df.Dispose();
567.             }
568.
569.             if (cf != null)
570.             {
571.                 cf.Dispose();
572.             }
573.         }
574.
575.
576.         //    // Process Depth
577.         //    FrameDescription dFrameDescription =
depthFrame.FrameDescription;
578.         }
579.         catch (Exception ex)
580.         {
581.             //nothing

```

```

582.         Debug.WriteLine(ex);
583.         if (df != null)
584.         {
585.             df.Dispose();
586.         }
587.
588.         if (cf != null)
589.         {
590.             cf.Dispose();
591.         }
592.     }
593. finally
594. {
595.     if (isBitmapLocked)
596.     {
597.         // this.bitmap.Unlock();
598.     }
599.
600.     if (df != null)
601.     {
602.         df.Dispose();
603.     }
604.
605.     if (cf != null)
606.     {
607.         cf.Dispose();
608.     }
609.
610.
611.     }
612.
613.     } // end if TouchPause
614.
615. }
616.
617.
618. private void RenderDepthPixels()
619. {
620.     //write image to screen - if it's called
621.     this.depthBitmap.WritePixels(
622.         new Int32Rect(0, 0, this.depthBitmap.PixelWidth,
this.depthBitmap.PixelHeight),
623.         depthPixels,
624.         this.depthBitmap.PixelWidth,
625.         0);
626.
627.
628.         //displayMe
629.         _bitmap.Lock();
630.         Marshal.Copy(_displayPixels, 0, _bitmap.BackBuffer,
_displayPixels.Length);
631.         _bitmap.AddDirtyRect(new Int32Rect(0, 0, depthWidth,
depthHeight));
632.         _bitmap.Unlock();

```

```

633.
634.
635.
636.         if (_popup != null)
637.         {
638.             _popup.startImage.Source = this.depthBitmap;
639.             _popup.colorImage.Source = this._bitmap;
640.         }
641.     }
642.
643.         /// <summary>
644.         /// Directly accesses the underlying image buffer of the
DepthFrame to
645.         /// create a displayable bitmap.
646.         /// This function requires the /unsafe compiler option as we
make use of direct
647.         /// access to the native memory pointed to by the
depthFrameData pointer.
648.         /// </summary>
649.         /// <param name="depthFrameData">Pointer to the DepthFrame
image data</param>
650.         /// <param name="depthFrameDataSize">Size of the DepthFrame
image data</param>
651.         /// <param name="minDepth">The minimum reliable depth value
for the frame</param>
652.         /// <param name="maxDepth">The maximum reliable depth value
for the frame</param>
653.         private unsafe void ProcessDepthFrameData(IntPtr
depthFrameData, uint depthFrameDataSize, ushort minDepth, ushort
maxDepth)
654.         {
655.             // depth frame data is a 16 bit value
656.             ushort* frameData = (ushort*)depthFrameData;
657.
658.             // convert depth to a visual representation
659.             for (int i = 0; i < (int)(depthFrameDataSize /
this.depthFrameDescription.BytesPerPixel); ++i)
660.             {
661.                 // Get the depth for this pixel
662.                 ushort depth = frameData[i];
663.
664.                 // To convert to a byte, we're mapping the depth
value to the byte range.
665.                 // Values outside the reliable depth range are
mapped to 0 (black).
666.                 //depthPixels[i] = (byte)(depth >= minDepth && depth
<= maxDepth ? (depth / MapDepthToByte) : 0);
667.                 //this.depthPixels[i] = (byte)(depth);
668.
669.                 depthPixels[i] = (byte)(depth >= minDepth && depth
<= maxDepth ? depth : 0);
670.
671.             }
672.

```

```

673.
674.
675.         }
676.
677.         //listens to key press on keyboard. - for testing.
678.         private void OnWindowKeyUp(object sender, KeyEventArgs e)
679.         {
680.             switch (e.Key)
681.             {
682.                 case Key.F1:
683.                     var ucs =
LogicalTreeHelper.FindLogicalNode(mainWindow, "uc" +
(tabControl.SelectedIndex + 1).ToString()) as
NonAffineTransformControl;
684.                     ucs.VirtualWispClick("aaa-111-bbb");
685.                     break;
686.                     //toggle fullscreen
687.                 case Key.F11:
688.                     var uc =
LogicalTreeHelper.FindLogicalNode(mainWindow, "uc" +
(tabControl.SelectedIndex + 1).ToString()) as
NonAffineTransformControl;
689.                     var tab =
LogicalTreeHelper.FindLogicalNode(mainWindow, "tabGrid" +
(tabControl.SelectedIndex + 1).ToString()) as Grid;
690.
691.                     if (uc != null && tab != null)
692.                     {
693.                         if (mainWindow.WindowState ==
System.Windows.WindowState.Maximized)
694.                         {
695.                             mainWindow.WindowState =
System.Windows.WindowState.Normal;
696.                             mainWindow.WindowStyle =
System.Windows.WindowStyle.SingleBorderWindow;
697.
698.                             rootGrid.Children.Remove(uc);
699.                             tab.Children.Add(uc);
700.                             tabControl.Visibility =
Visibility.Visible;
701.
702.                         }
703.                         else
704.                         {
705.                             mainWindow.WindowState =
System.Windows.WindowState.Maximized;
706.                             mainWindow.WindowStyle =
System.Windows.WindowStyle.None;
707.
708.                             tab.Children.Remove(uc);
709.                             rootGrid.Children.Add(uc);
710.                             tabControl.Visibility =
Visibility.Hidden;
711.

```

```

712.         }
713.     }
714.     break;
715.     case Key.F2: //TEMP functional to reset XML to test
        setup
716.         Wisp[] wisps = new Wisp[] {
717.             new Wisp()
718.             {
719.                 WispID = "aaa-111-bbb",
720.                 Name = "ccc",
721.                 WispPolygon = new
WispPolygon()
722.                 {
723.                     Content = "gif.gif",
724.                     ClickContent =
"gif.gif",
725.                     CornerPoints = new
PointCollection(new Point[] {
726.                         new Point(0, 0),
//Coordinates of a top-left corner
727.                         new Point(0,
100), //Coordinates of a bottom-left corner
728.                         new Point(100,
0), //Coordinates of a top-right corner
729.                         new Point(100,
100) //Coordinates of a bottom-right corner
730.                     }},
731.                     ContentType =
ContentType.Gif,
732.                     FrameRate = 0,
733.                     Loop =
LoopType.Forever,
734.                     ZOrder = 1
735.                 },
736.                 ContentPolygon = new
ContentPolygon()
737.                 {
738.                     Content =
"PetzoldTattoo.jpg",
739.                     CornerPoints = new
PointCollection(new Point[] {
740.                         new Point(100,
0), //Coordinates of a top-left corner
741.                         new Point(100,
100), //Coordinates of a bottom-left corner
742.                         new Point(200,
0), //Coordinates of a top-right corner
743.                         new Point(200,
100) //Coordinates of a bottom-right corner
744.                     }},
745.                     ContentType =
ContentType.Image,
746.                     FrameRate = 0,
747.                     Loop = (LoopType)2,

```

```

748.                                     ZOrder = 0,
749.                                     ContentOrientation =
750.                                     0,
751.                                     Timeout = 3,
752.                                     }
753.                                     },
754.                                     new Wisp()
755.                                     {
756.                                         WispID = "aaa-111-bbb",
757.                                         Name = "ccc",
758.                                         WispPolygon = new
759.                                             WispPolygon()
760.                                             {
761.                                                 Content = "vid1.wmv",
762.                                                 ClickContent = null,
763.                                                 CornerPoints = new
764.                                                     PointCollection(new Point[] {
765.                                                         new Point(0,
766.                                                             100), //Coordinates of a top-left corner
767.                                                         new Point(0,
768.                                                             200), //Coordinates of a bottom-left corner
769.                                                         new Point(100,
770.                                                             100), //Coordinates of a top-right corner
771.                                                         new Point(100,
772.                                                             200) //Coordinates of a bottom-right corner
773.                                                     }),
774.                                                 ContentType =
775.                                                     ContentType.Video,
776.                                                 FrameRate = 0,
777.                                                 Loop = (LoopType)2,
778.                                                 ZOrder = 4
779.                                             },
780.                                         ContentPolygon = new
781.                                             ContentPolygon()
782.                                             {
783.                                                 Content =
784.                                                     "PetzoldTattoo.jpg",
785.                                                 CornerPoints = new
786.                                                     PointCollection(new Point[] {
787.                                                         new Point(100,
788.                                                             0), //Coordinates of a top-left corner
789.                                                         new Point(100,
790.                                                             100), //Coordinates of a bottom-left corner
791.                                                         new Point(200,
792.                                                             0), //Coordinates of a top-right corner
793.                                                         new Point(200,
794.                                                             100) //Coordinates of a bottom-right corner
795.                                                     }),
796.                                                 ContentType =
797.                                                     ContentType.Image,
798.                                                 FrameRate = 0,
799.                                                 Loop = (LoopType)2,
800.                                                 ZOrder = 0,

```

```

785.                                     ContentOrientation =
      0,
786.                                     Timeout = 3,
787.                                     }
788.                               },
789.                               new Wisp()
790.                               {
791.                                   WispID = "aaa-222-bbb",
792.                                   Name = "ccccxzcxxxxcz",
793.                                   WispPolygon = new
WispPolygon()
794.                                   {
795.                                       Content =
"wisp_png_sequence",
796.                                       ClickContent = null,
797.                                       CornerPoints = new
PointCollection(new Point[] {
798.                                           new Point(0,
200), //Coordinates of a top-left corner
799.                                           new Point(0,
300), //Coordinates of a bottom-left corner
800.                                           new Point(100,
200), //Coordinates of a top-right corner
801.                                           new Point(100,
300) //Coordinates of a bottom-right corner
802.                                       })),
803.                                       ContentType =
ContentType.PngSeq,
804.                                       FrameRate = 60,
805.                                       Loop = (LoopType)2,
806.                                       ZOrder = 2
807.                                   },
808.                                   ContentPolygon = new
ContentPolygon()
809.                                   {
810.                                       Content =
"PetzoldTattoo.jpg",
811.                                       CornerPoints = new
PointCollection(new Point[] {
812.                                           new Point(100,
0), //Coordinates of a top-left corner
813.                                           new Point(100,
100), //Coordinates of a bottom-left corner
814.                                           new Point(200,
0), //Coordinates of a top-right corner
815.                                           new Point(200,
100) //Coordinates of a bottom-right corner
816.                                       })),
817.                                       ContentType =
ContentType.Image,
818.                                       FrameRate = 0,
819.                                       Loop = (LoopType)2,
820.                                       ZOrder = 0,

```

```

821.                                     ContentOrientation =
      0,
822.                                     Timeout = 3,
823.                                     }
824.                                     },
825.                                     new Wisp()
826.                                     {
827.                                         WispID = "aaa-111-bbb",
828.                                         Name = "cccwqqweqw",
829.                                         WispPolygon = new
WispPolygon()
830.                                     {
831.                                         Content =
      "wisp_png_sequence2",
832.                                         ClickContent = null,
833.                                         CornerPoints = new
      PointCollection(new Point[] {
834.                                             new Point(0,
      300), //Coordinates of a top-left corner
835.                                             new Point(0,
      400), //Coordinates of a bottom-left corner
836.                                             new Point(100,
      300), //Coordinates of a top-right corner
837.                                             new Point(100,
      400) //Coordinates of a bottom-right corner
838.                                         }),
839.                                         ContentType =
      ContentType.Text,
840.                                         FrameRate = 60,
841.                                         Loop = (LoopType)2,
842.                                         ZOrder = 5
843.                                     },
844.                                     ContentPolygon = new
      ContentPolygon()
845.                                     {
846.                                         Content =
      "PetzoldTattoo.jpg",
847.                                         CornerPoints = new
      PointCollection(new Point[] {
848.                                             new Point(100,
      0), //Coordinates of a top-left corner
849.                                             new Point(100,
      100), //Coordinates of a bottom-left corner
850.                                             new Point(200,
      0), //Coordinates of a top-right corner
851.                                             new Point(200,
      100) //Coordinates of a bottom-right corner
852.                                         }),
853.                                         ContentType =
      ContentType.Image,
854.                                         FrameRate = 0,
855.                                         Loop = (LoopType)2,
856.                                         ZOrder = 0,

```

```

857.                                     ContentOrientation =
      0,
858.                                     Timeout = 3,
859.                                     }
860.                                     },
861.                                     new Wisp()
862.                                     {
863.                                         WispID = "aaa-111-bbb",
864.                                         Name = "cccdas",
865.                                         WispPolygon = new
WispPolygon()
866.                                     {
867.                                         Content =
      "PetzoldTattoo.jpg",
868.                                         ClickContent =
      "rotate.png",
869.                                         CornerPoints = new
PointCollection(new Point[] {
870.                                             new Point(0,
400), //Coordinates of a top-left corner
871.                                             new Point(0,
500), //Coordinates of a bottom-left corner
872.                                             new Point(100,
400), //Coordinates of a top-right corner
873.                                             new Point(100,
500) //Coordinates of a bottom-right corner
874.                                         }),
875.                                         ContentType =
      ContentType.Image,
876.                                         FrameRate = 60,
877.                                         Loop = (LoopType)2,
878.                                         ZOrder = 0
879.                                     },
880.                                         ContentPolygon = new
ContentPolygon()
881.                                     {
882.                                         Content =
      "PetzoldTattoo.jpg",
883.                                         CornerPoints = new
PointCollection(new Point[] {
884.                                             new Point(100,
0), //Coordinates of a top-left corner
885.                                             new Point(100,
100), //Coordinates of a bottom-left corner
886.                                             new Point(200,
0), //Coordinates of a top-right corner
887.                                             new Point(200,
100) //Coordinates of a bottom-right corner
888.                                         }),
889.                                         ContentType =
      ContentType.Image,
890.                                         FrameRate = 0,
891.                                         Loop = (LoopType)2,
892.                                         ZOrder = 0,

```

```

893.                                     ContentOrientation =
      0,
894.                                     Timeout = 3,
895.                                     }
896.                                     }
897.                                     };
898.                                     //ucl.Wisps = wisps;
899.
900.                                     using (var fs = File.Create("test.xml"))
901.                                     {
902.                                         XmlSerializer xs = new
      XmlSerializer(typeof(Wisp[]), new Type[] {
903.                                             typeof(Wisp),
904.                                             typeof(ContentPolygon),
905.                                             typeof(WispPolygon),
906.                                             typeof(Polygon),
907.                                             typeof(ContentType),
908.                                             typeof(LoopType)
909.                                         });
910.                                         xs.Serialize(fs, wisps);
911.                                     }
912.
913.                                     break;
914.                                 }
915.                            }
916.                            private void OnViewerMouseUp(object sender,
      MouseButtonEventArgs e)
917.                            {
918.                                enterViewerMode = true;
919.                                enterTab2 = false;
920.                            }
921.
922.                            private void OnSetupMouseUp(object sender,
      MouseButtonEventArgs e)
923.                            {
924.                                enterViewerMode = false;
925.                                enterTab2 = false;
926.                            }
927.                            private void OnTab2MouseUp(object sender,
      MouseButtonEventArgs e)
928.                            {
929.                                enterTab2 = true;
930.                                enterViewerMode = false;
931.
932.                                //if popup is closed
933.                                if (_popup != null)
934.                                {
935.                                    _popup.startImage.Source = new BitmapImage(new
      Uri(@"pack://siteoforigin:,,,/Spaceeblk.jpg"));
936.                                }
937.                                if (_popup == null)
938.                                {
939.                                    //creating a popup
940.                                    _popup = new PopupWindow();

```

```

941.         _popup.Closed += OnPopupClosed;
942.
943.
944.         //finding out screen configuration
945.         var allScreens =
System.Windows.Forms.Screen.AllScreens;
946.         //by default destination screen is the first
947.         System.Windows.Forms.Screen destScreen =
allScreens[0];
948.         //if there are more than one screen
949.         if (allScreens.Length > 1)
950.         {
951.             //finding out current screen
952.             var currentScreen =
System.Windows.Forms.Screen.FromPoint(new
System.Drawing.Point((int)Left, (int)Top));
953.             var screenIndex = Array.IndexOf(allScreens,
currentScreen);
954.             //destination screen is next
955.             destScreen = allScreens[(screenIndex + 1) %
allScreens.Length];
956.         }
957.         //placing popup in the middle of destination screen
958.         _popup.Left = destScreen.WorkingArea.Left +
(destScreen.WorkingArea.Width - _popup.Width) / 2;
959.         _popup.Top = destScreen.WorkingArea.Top +
(destScreen.WorkingArea.Height - _popup.Height) / 2;
960.
961.         //copy wisps to popup
962.         _popup.NonAffineTransformControl.Wisps = uc2.Wisps;
963.
964.         //showing popup
965.         _popup.Show();
966.     }
967.     else //else focusing it
968.     {
969.         _popup.Focus();
970.     }
971. }
972.
973. /// <summary>
974. /// Handles popup closed event. Removes all handlers and
links.
975. /// </summary>
976. /// <param name="sender"></param>
977. /// <param name="e"></param>
978. private void OnPopupClosed(object sender, EventArgs e)
979. {
980.     _popup.Closed -= OnPopupClosed;
981.     _popup = null;
982. }
983.
984. /// <summary>
985. /// Handles popup NonAffineTransformControl loaded event.

```

```

986.          /// </summary>
987.          /// <param name="sender"></param>
988.          /// <param name="e"></param>
989.
990.
991.
992.
993.
994.
995.          private void OnMainWindowLoaded(object sender,
RoutedEventArgs e)
996.          {
997.              UpdateCornerPoints();
998.          }
999.
1000.         private void OnMainWindowLocationChanged(object sender,
EventArgs e)
1001.         {
1002.             UpdateCornerPoints();
1003.         }
1004.
1005.         private void OnMainWindowSizeChanged(object sender,
SizeChangedEventArgs e)
1006.         {
1007.             UpdateCornerPoints();
1008.         }
1009.
1010.         private void UpdateCornerPoints()
1011.         {
1012.             _topLeft = PointToScreen(new Point());
1013.             _bottomRight = new Point(_topLeft.X +
mainWindow.ActualWidth, _topLeft.Y + mainWindow.ActualHeight);
1014.         }
1015.
1016.
1017.
1018.         private PopupWindow _popup;
1019.         private Point _topLeft;
1020.         private Point _bottomRight;
1021.
1022.
1023.         //loads the system in viewer mode and in full screen. - If
there is a system crash, the monoter system will detect it and
relaunch the program in this state.
1024.         public void startAppinFullView()
1025.         {
1026.             //wait until the program loads
1027.             int ct = 0;
1028.             DispatcherTimer switchTabTimer = new DispatcherTimer();
1029.             switchTabTimer.Interval = TimeSpan.FromSeconds(1);
1030.             switchTabTimer.Tick += (object timerSender, EventArgs
timerE) =>
1031.             {
1032.                 ct++;

```

```

1033.          //switchTabTimer.Stop();
1034.          if (ct > 0)
1035.          {
1036.              //viewer tab
1037.              tabControl.SelectedIndex = 2;
1038.              switchTabTimer.Stop();
1039.              switchTabTimer = null;
1040.              // System.Windows.MessageBox.Show("PREP");
1041.              var uc =
LogicalTreeHelper.FindLogicalNode(mainWindow, "uc" +
(tabControl.SelectedIndex + 1).ToString()) as
NonAffineTransformControl;
1042.              var tab =
LogicalTreeHelper.FindLogicalNode(mainWindow, "tabGrid" +
(tabControl.SelectedIndex + 1).ToString()) as Grid;
1043.              if (uc != null && tab != null)
1044.              {
1045.
1046.                  mainWindow.WindowState =
System.Windows.WindowState.Maximized;
1047.                  mainWindow.WindowStyle =
System.Windows.WindowStyle.None;
1048.
1049.                  tab.Children.Remove(uc);
1050.                  rootGrid.Children.Add(uc);
1051.                  tabControl.Visibility = Visibility.Hidden;
1052.
1053.                  enterViewerMode = true;
1054.                  enterTab2 = false;
1055.                  //System.Windows.MessageBox.Show("IT
WORKS");
1056.
1057.                  //CAUSE ERROR TO TEST APPLICATION CRASH
1058.                  // int t = 0 / (ct - 1);
1059.
1060.              }
1061.          }
1062.      };
1063.      switchTabTimer.Start();
1064.
1065.
1066.  }
1067.
1068.      //DEPTHS XML
1069.      private void SaveWispsToXML()
1070.      {
1071.          using (var fs =
System.IO.File.Create(_WispCoveredFrameItem_XMLFile))
1072.          {
1073.              XmlSerializer xs = new
XmlSerializer(typeof(List<WispCoveredFrameItem>), new Type[] {
typeof(List<WispCoveredFrameItem>)
1074.                  });
1075.              xs.Serialize(fs, WispViewTabList);
1076.

```

```

1077.
1078.             fs.Close();
1079.         }
1080.     }
1081.     private void LoadWispsFromXML()
1082.     {
1083.         if (!string.IsNullOrEmpty(_WispCoveredFrameItem_XMLFile)
1084.         && System.IO.File.Exists(_WispCoveredFrameItem_XMLFile))
1085.         {
1086.             try
1087.             {
1088.                 using (var fs =
1089.                 System.IO.File.OpenRead(_WispCoveredFrameItem_XMLFile))
1090.                 {
1091.                     XmlSerializer xs = new
1092.                     XmlSerializer(typeof(List<WispCoveredFrameItem>), new Type[] {
1093.                         typeof(WispCoveredFrameItem)
1094.                     });
1095.                     WispViewTabList =
1096.                     (List<WispCoveredFrameItem>)xs.Deserialize(fs);
1097.                 }
1098.             }
1099.             catch { }
1100.         }
1101.     }
1102.
1103.     private void SaveWispDepthsToXML()
1104.     {
1105.         using (var fs =
1106.         System.IO.File.Create(_WispDepths_XMLFile))
1107.         {
1108.             XmlSerializer xs = new
1109.             XmlSerializer(typeof(List<WispInteractiveData>), new Type[] {
1110.                 typeof(List<WispInteractiveData>)
1111.             });
1112.             xs.Serialize(fs, wispDepths);
1113.             fs.Close();
1114.         }
1115.     }
1116.
1117.     private void LoadWispDepthsFromXML()
1118.     {
1119.         if (!string.IsNullOrEmpty(_WispDepths_XMLFile) &&
1120.         System.IO.File.Exists(_WispDepths_XMLFile))
1121.         {
1122.             try
1123.             {
1124.                 using (var fs =
1125.                 System.IO.File.OpenRead(_WispDepths_XMLFile))
1126.                 {
1127.                     XmlSerializer xs = new
1128.                     XmlSerializer(typeof(List<WispInteractiveData>), new Type[] {

```

```
1122.                                     typeof(WispInteractiveData)
1123.                                     });
1124.                                     wispDepths =
(List<WispInteractiveData>)xs.Deserialize(fs);
1125.                                     }
1126.                                     }
1127.                                     catch { }
1128.                                     }
1129.                                     }
1130.
1131.
1132.     }
1133.
1134.
1135. }
1136.
```

```

1.    using System;
2.    using System.Collections.Generic;
3.    using System.Linq;
4.    using System.Text;
5.    using System.Threading.Tasks;
6.    using System.Windows;
7.    using System.Windows.Controls;
8.    using System.Windows.Data;
9.    using System.Windows.Documents;
10.   using System.Windows.Input;
11.   using System.Windows.Media;
12.   using System.Windows.Media.Imaging;
13.   using System.Windows.Media.Media3D;
14.   using System.Windows.Navigation;
15.   using System.Windows.Shapes;
16.   using System.Windows.Threading;
17.   using System.Xml.Serialization;
18.
19.   namespace NonAffineTransform
20.   {
21.       /// <summary>
22.       /// Interaction logic for NonAffineTransformControl.xaml
23.       /// </summary>
24.       public partial class NonAffineTransformControl : UserControl
25.       {
26.           private class RenderedPolygon
27.           {
28.               /// <summary>
29.               /// Rendering planes (meshes)
30.               /// </summary>
31.               public MeshGeometry3D Mesh;
32.
33.               /// <summary>
34.               /// Transforming matrices
35.               /// </summary>
36.               public MatrixTransform3D MatrixTransform;
37.
38.               /// <summary>
39.               /// Corner points of meshes
40.               /// </summary>
41.               public Point3D[] PointsTransformed;
42.
43.               /// <summary>
44.               /// Collection of WISP outlines
45.               /// </summary>
46.               public PathFigure Outline;
47.
48.               /// <summary>
49.               /// Collection of WISP outline paths
50.               /// </summary>
51.               public Path OutlinePath;
52.
53.               /// <summary>
54.               /// Connected control

```

```

55.         /// </summary>
56.         public FrameworkElement Control;
57.
58.         /// <summary>
59.         /// Collection of WISP materials
60.         /// </summary>
61.         public DiffuseMaterial Material;
62.
63.         /// <summary>
64.         /// Polygon that is being rendered
65.         /// </summary>
66.         public Polygon Parent;
67.     }
68.
69.     private class RenderedDefaultPolygon : RenderedPolygon
70.     {
71.         /// <summary>
72.         /// ID of default polygon
73.         /// </summary>
74.         public string ID;
75.
76.         /// <summary>
77.         /// How many times it's requested to hide
78.         /// </summary>
79.         public int HideCount;
80.     }
81.
82.     private class RenderedWisp
83.     {
84.         /// <summary>
85.         /// Rendered Wisp polygon
86.         /// </summary>
87.         public RenderedPolygon WispPolygon;
88.
89.         /// <summary>
90.         /// Rendered content polygon
91.         /// </summary>
92.         public RenderedPolygon ContentPolygon;
93.
94.         /// <summary>
95.         /// Wisp that is being rendered
96.         /// </summary>
97.         public Wisp Parent;
98.
99.         /// <summary>
100.        /// Determines whether content polygons is shown
101.        /// </summary>
102.        public bool ContentPolygonShown;
103.
104.        /// <summary>
105.        /// Backup of content polygon to restore after making
106.        unwanted changes
107.        /// </summary>
108.        public ContentPolygon ContentPolygonBackup;

```

```

108.
109.         /// <summary>
110.         /// Timer which hides content polygon on timeout
111.         /// </summary>
112.         public DispatcherTimer HideContentPolygonTimer;
113.
114.         /// <summary>
115.         /// Media player which plays click sounds
116.         /// </summary>
117.         public MediaPlayer MediaPlayer;
118.     }
119.
120.     /// <summary>
121.     /// Shows priority of mesh dragging over vertex dragging.
122.     /// Set to 1 to disable vertex dragging.
123.     /// Set to 0 to disable mesh dragging.
124.     /// Balanced values are 0.6-0.7 .
125.     /// </summary>
126.     private const double MeshDraggingWeight = 0.65;
127.
128.     /// <summary>
129.     /// Shows how far next rendering plane would be placed
130.     /// </summary>
131.     private const double ZIncrement = -10.0;
132.
133.     private static readonly Point3D[] DefaultCornerPoints = new
Point3D[] {
134.         new Point3D(-0.5, 0.5, 0),
135.         new Point3D(-0.5, -0.5, 0),
136.         new Point3D(0.5, 0.5, 0),
137.         new Point3D(0.5, -0.5, 0)
138.     };
139.
140.     public static readonly DependencyProperty WispsProperty;
141.     public static readonly DependencyProperty
DefaultPolygonProperty;
142.     public static readonly DependencyProperty
StorageFileProperty;
143.     public static readonly DependencyProperty
ShowContentProperty;
144.     public static readonly DependencyProperty
ShowOutlinesProperty;
145.     public static readonly DependencyProperty ReadOnlyProperty;
146.     public static readonly DependencyProperty PlaySoundProperty;
147.
148.     static NonAffineTransformControl()
149.     {
150.         WispsProperty = DependencyProperty.Register("Wisps",
typeof(Wisp[]), typeof(NonAffineTransformControl));
151.         DefaultPolygonProperty =
DependencyProperty.Register("DefaultPolygons",
typeof(DefaultPolygon[]), typeof(NonAffineTransformControl));

```

```

152.         StorageFileProperty =
DependencyProperty.Register("Storage", typeof(string),
typeof(NonAffineTransformControl));
153.         ShowContentProperty =
DependencyProperty.Register("ShowContent", typeof(bool),
typeof(NonAffineTransformControl), new PropertyMetadata(true));
154.         ShowOutlinesProperty =
DependencyProperty.Register("ShowOutlines", typeof(bool),
typeof(NonAffineTransformControl), new PropertyMetadata(true));
155.         ReadOnlyProperty =
DependencyProperty.Register("ReadOnly", typeof(bool),
typeof(NonAffineTransformControl), new PropertyMetadata(false));
156.         PlaySoundProperty =
DependencyProperty.Register("PlaySound", typeof(bool),
typeof(NonAffineTransformControl), new PropertyMetadata(false));
157.     }
158.
159.     public NonAffineTransformControl()
160.     {
161.         InitializeComponent();
162.
163.         Loaded += OnNonAffineTransformControlLoaded;
164.         SizeChanged += OnNonAffineTransformControlSizeChanged;
165.     }
166.
167.     private void OnHideContentPolygonTimerTick(object sender,
EventArgs e)
168.     {
169.         var fe = (sender as DispatcherTimer);
170.         if (fe == null)
171.             return;
172.
173.         var rw = fe.Tag as RenderedWisp;
174.         if (rw == null)
175.             return;
176.
177.         HideContentPolygon(rw);
178.     }
179.
180.     /// <summary>
181.     /// Gets or sets collection of polygons which is used to
display images/videos/etc.
182.     /// </summary>
183.     public Wisp[] Wisps
184.     {
185.         get { return (Wisp[])GetValue(WispsProperty); }
186.         set
187.         {
188.             SetValue(WispsProperty, value);
189.             if (!_internalChange && _loaded)
190.             {
191.                 Initialize();
192.                 UpdateCornerPointPositions();
193.             }

```

```

194.         }
195.     }
196.
197.         /// <summary>
198.         /// Gets or sets collection of polygons which is used to
display images/videos/etc.
199.         /// </summary>
200.         public DefaultPolygon[] DefaultPolygons
201.         {
202.             get { return
(DefaultPolygon[]) GetValue(DefaultPolygonProperty); }
203.             set
204.             {
205.                 SetValue(DefaultPolygonProperty, value);
206.                 //if (!_internalChange && _loaded)
207.                 //{
208.                     //    Initialize();
209.                     //    UpdateCornerPointPositions();
210.                 //}
211.             }
212.         }
213.
214.         /// <summary>
215.         /// Gets or sets path to xml file where control autosaves
and loads Wisps
216.         /// </summary>
217.         public string StorageFile
218.         {
219.             get { return (string) GetValue(StorageFileProperty); }
220.             set
221.             {
222.                 SetValue(StorageFileProperty, value);
223.             }
224.         }
225.
226.         /// <summary>
227.         /// Gets or sets value indication whether content of wisp
polygons is shown.
228.         /// </summary>
229.         public bool ShowContent
230.         {
231.             get { return (bool) GetValue(ShowContentProperty); }
232.             set
233.             {
234.                 SetValue(ShowContentProperty, value);
235.                 viewport3d.Visibility = value ? Visibility.Visible :
Visibility.Hidden;
236.             }
237.         }
238.
239.         /// <summary>
240.         /// Gets or sets value indicating whether outlines and
controls of content polygons are shown.
241.         /// </summary>

```

```

242.         public bool ShowOutlines
243.         {
244.             get { return (bool)GetValue(ShowOutlinesProperty); }
245.             set
246.             {
247.                 SetValue(ShowOutlinesProperty, value);
248.                 overlay.Visibility = value ? Visibility.Visible :
Visibility.Hidden;
249.             }
250.         }
251.
252.         /// <summary>
253.         /// Gets or sets value indicating whether user able to move
polygons.
254.         /// </summary>
255.         public bool ReadOnly
256.         {
257.             get { return (bool)GetValue(ReadOnlyProperty); }
258.             set
259.             {
260.                 SetValue(ReadOnlyProperty, value);
261.             }
262.         }
263.
264.         /// <summary>
265.         /// Gets or sets value indicating whether control should
play click sounds.
266.         /// </summary>
267.         public bool PlaySound
268.         {
269.             get { return (bool)GetValue(PlaySoundProperty); }
270.             set
271.             {
272.                 SetValue(PlaySoundProperty, value);
273.             }
274.         }
275.
276.         public void VirtualWispClick(String wispID)
277.         {
278.             for (int i = 0; i < _renderedWisps.Length; i++)
279.             {
280.                 if (_renderedWisps[i].Parent.WispID == wispID)
281.                 {
282.                     ShowContentPolygon(_renderedWisps[i]);
283.                     break;
284.                 }
285.             }
286.         }
287.
288.         /// <summary>
289.         /// Returns a Wisp at specified location
290.         /// </summary>
291.         /// <param name="point"></param>
292.         /// <returns></returns>

```

```

293.         public Wisp GetWispAtPoint(Point point)
294.         {
295.             var wisps = Wisps;
296.             if (wisps == null)
297.                 return null;
298.
299.             var resultMesh = GetMeshAtPoint(point);
300.             if (resultMesh == null)
301.                 return null;
302.
303.             var meshIndex = Array.IndexOf(_renderedWisps.Select(rw
=> rw.WispPolygon.Mesh).ToArray(), resultMesh.MeshHit);
304.             if (meshIndex >= 0 && meshIndex < wisps.Length)
305.             {
306.                 var resultWisp = wisps[meshIndex];
307.
308.                 return resultWisp;
309.             }
310.
311.             return null;
312.         }
313.
314.         /// <summary>
315.         /// Returns a Wisp at specified location
316.         /// </summary>
317.         /// <param name="point"></param>
318.         /// <returns></returns>
319.         public Wisp GetWispAtPoint(double x, double y)
320.         {
321.             return GetWispAtPoint(new Point(x, y));
322.         }
323.
324.         /// <summary>
325.         /// Returns a Wisp ID at specified location
326.         /// </summary>
327.         /// <param name="point"></param>
328.         /// <returns></returns>
329.         public string GetWispIDAtPoint(Point point)
330.         {
331.             var wisp = GetWispAtPoint(point);
332.             return wisp != null ? wisp.WispID : null;
333.         }
334.
335.         /// <summary>
336.         /// Returns a Wisp ID at specified location
337.         /// </summary>
338.         /// <param name="point"></param>
339.         /// <returns></returns>
340.         public string GetWispIDAtPoint(double x, double y)
341.         {
342.             return GetWispIDAtPoint(new Point(x, y));
343.         }
344.

```

```

345.         protected override void
OnMouseLeftButtonDown(MouseButtonEventArgs args)
346.     {
347.         Point pt = args.GetPosition(viewport3d);
348.
349.         var resultMesh = GetMeshAtPoint(pt);
350.         if (resultMesh == null)
351.             return;
352.
353.         var result = _renderedWisps.FirstOrDefault(rw =>
rw.WispPolygon.Mesh == resultMesh.MeshHit);
354.         if (result != null)
355.         {
356.             _draggingWisp = result;
357.             _draggingPolygon = result.WispPolygon;
358.         }
359.         else
360.         {
361.             result = _renderedWisps.FirstOrDefault(rw =>
rw.ContentPolygonShown && rw.ContentPolygon.Mesh ==
resultMesh.MeshHit);
362.             if (result != null)
363.             {
364.                 _draggingWisp = result;
365.                 _draggingPolygon = result.ContentPolygon;
366.                 _isDraggingContent = true;
367.             }
368.             else
369.             {
370.                 var defaultPolygon =
_renderedDefaultPolygons.FirstOrDefault(rdp => rdp.Mesh ==
resultMesh.MeshHit);
371.                 if (defaultPolygon != null)
372.                 {
373.                     _draggingPolygon = defaultPolygon;
374.                 }
375.                 else
376.                 {
377.                     return;
378.                 }
379.             }
380.         }
381.
382.         if (!ReadOnly && _renderedWisps.Any(rw =>
rw.ContentPolygonShown && rw.ContentPolygon != _draggingPolygon &&
rw.WispPolygon != _draggingPolygon))
383.         {
384.             _draggingWisp = null;
385.             _draggingPolygon = null;
386.             _isDragging = false;
387.             _isDraggingContent = false;
388.             return;
389.         }
390.

```

```

391.         var diff12 = resultMesh.VertexWeight1 -
resultMesh.VertexWeight2;
392.         var diff13 = resultMesh.VertexWeight1 -
resultMesh.VertexWeight3;
393.         var diff23 = resultMesh.VertexWeight2 -
resultMesh.VertexWeight3;
394.
395.         // Determine if the mouse within central part of mesh.
396.         if (Math.Abs(diff12) < MeshDraggingWeight &&
Math.Abs(diff13) < MeshDraggingWeight && Math.Abs(diff23) <
MeshDraggingWeight)
397.         {
398.             _draggingVertexIndex = -1;
399.         }
400.         // Determine which vertex the mouse is closest to.
401.         else if (diff12 > 0)
402.         {
403.             _draggingVertexIndex = diff13 > 0 ?
resultMesh.VertexIndex1 : resultMesh.VertexIndex3;
404.         }
405.         else
406.         {
407.             _draggingVertexIndex = diff23 > 0 ?
resultMesh.VertexIndex2 : resultMesh.VertexIndex3;
408.         }
409.
410.         _startDraggingMousePoint = pt;
411.         if (!ReadOnly)
412.         {
413.             // Get current transformed points.
414.             for (int i = 0; i < 4; i++)
415.             {
416.                 _draggingPolygon.PointsTransformed[i] =
_draggingPolygon.MatrixTransform.Matrix.Transform(_draggingPolygon.M
esh.Positions[i]);
417.             }
418.             if (_draggingVertexIndex >= 0)
419.             {
420.
//_draggingPolygon.PointsTransformed[_draggingVertexIndex] =
Simple2Dto3D(viewport3d, pt,
_draggingPolygon.PointsTransformed[_draggingVertexIndex].Z);
421.                 //_draggingPolygon.MatrixTransform.Matrix =
CalculateNonAffineTransform(_draggingPolygon.PointsTransformed);
422.             }
423.             else
424.             {
425.                 //saving start coordinates for mesh dragging
426.                 _startDraggingPoint = Simple2Dto3D(viewport3d,
pt);
427.                 _startDraggingMeshPoints = new Point3D[4];
428.
Array.ConstrainedCopy(_draggingPolygon.PointsTransformed, 0,
_startDraggingMeshPoints, 0, 4);

```

```

429.         }
430.     }
431.     //Commence dragging operation.
432.     _isDragging = true;
433.     CaptureMouse();
434.     //args.Handled = true;
435. }
436.
437.     protected override void OnMouseMove(MouseEventArgs args)
438.     {
439.         base.OnMouseMove(args);
440.
441.         //Determine if it is OK to move polygon or corner.
442.         if (!ReadOnly && _isDragging && _draggingPolygon != null
&& _renderedWisps.All(rw => !rw.ContentPolygonShown ||
rw.ContentPolygon == _draggingPolygon))
443.         {
444.             Point ptMouse = args.GetPosition(viewport3d);
445.
446.             // Vertex dragging
447.             if (_draggingVertexIndex >= 0)
448.             {
449.
450.                 _draggingPolygon.PointsTransformed[_draggingVertexIndex] =
Simple2Dto3D(viewport3d, ptMouse,
_draggingPolygon.PointsTransformed[_draggingVertexIndex].Z);
451.
452.                 _draggingPolygon.Parent.CornerPoints[_draggingVertexIndex] =
Simple3Dto2D(viewport3d,
_draggingPolygon.PointsTransformed[_draggingVertexIndex]);
453.             }
454.             // Mesh dragging
455.             else
456.             {
457.                 Point3D curPosition = Simple2Dto3D(viewport3d,
ptMouse);
458.                 Vector3D offset = curPosition -
_startDraggingPoint;
459.                 offset.Z = 0;
460.                 for (int i = 0; i < 4; i++)
461.                 {
462.                     _draggingPolygon.PointsTransformed[i] =
_startDraggingMeshPoints[i] + offset;
463.                     _draggingPolygon.Parent.CornerPoints[i] =
Simple3Dto2D(viewport3d, _draggingPolygon.PointsTransformed[i]);
464.                 }
465.             }
466.             //updating outline
467.             _draggingPolygon.Parent.UpdateOutline(_draggingPolygon.Outline);
468.

```

```

469.         Canvas.SetLeft(_draggingPolygon.Control,
        (_draggingPolygon.Parent.CornerPoints[1].X +
        _draggingPolygon.Parent.CornerPoints[3].X -
        _draggingPolygon.Control.ActualWidth) * 0.5);
470.         Canvas.SetTop(_draggingPolygon.Control,
        Math.Max(_draggingPolygon.Parent.CornerPoints[1].Y,
        _draggingPolygon.Parent.CornerPoints[3].Y));
471.
472.         //updating transformation matrix.
473.         _draggingPolygon.MatrixTransform.Matrix =
        CalculateNonAffineTransform(_draggingPolygon.PointsTransformed);
474.         //args.Handled = true;
475.     }
476. }
477.
478.     protected override void
        OnMouseLeftButtonUp(MouseButtonEventArgs args)
479.     {
480.         base.OnMouseUp(args);
481.
482.         if (_isDragging)
483.         {
484.             Point mousePoint = args.GetPosition(viewport3d);
485.
486.             //if polygon was not moved and user clicked on a
            right one, showing content polygon
487.             if (ShowContent && _draggingWisp != null &&
        _draggingWisp.WispPolygon == _draggingPolygon &&
488.             Math.Abs(mousePoint.X -
        _startDraggingMousePoint.X) < 2 && Math.Abs(mousePoint.Y -
        _startDraggingMousePoint.Y) < 2)
489.             {
490.                 if (!_draggingWisp.ContentPolygonShown)
491.                 {
492.                     ShowContentPolygon(_draggingWisp);
493.                 }
494.                 else //or hiding content polygon if it was shown
495.                 {
496.                     //reverting content polygon params, cause
                        Save Content button was not pressed
497.                     _draggingWisp.ContentPolygon.Parent =
        _draggingWisp.Parent.ContentPolygon =
        _draggingWisp.ContentPolygonBackup;
498.                     HideContentPolygon(_draggingWisp);
499.                 }
500.             }
501.             //else it was a drag operation and it's needed to
            save values to xml.
502.             else if (!ReadOnly && !_isDraggingContent)
503.             {
504.                 SaveData();
505.             }
506.
507.             _isDragging = false;

```

```

508.         _isDraggingContent = false;
509.         ReleaseMouseCapture();
510.         //args.Handled = true;
511.     }
512. }
513.
514. private RayMeshGeometry3DHitTestResult GetMeshAtPoint(Point
point)
515. {
516.     RayMeshGeometry3DHitTestResult resultMesh = null;
517.
518.     // Obtain the Visual3D objects under the mouse pointer.
519.     VisualTreeHelper.HitTest(viewport3d, null,
520.         new HitTestResultCallback((hitTestResult) =>
521.         {
522.             resultMesh = hitTestResult as
RayMeshGeometry3DHitTestResult;
523.
524.             //which is not an invisible content polygon or
something else
525.             if (resultMesh == null || (resultMesh.VisualHit
as ModelVisual3D) == null ||
526.                 Array.IndexOf(_renderedWisps.Select(rw =>
rw.WispPolygon.Mesh).ToArray(), resultMesh.MeshHit) < 0 &&
527.                 Array.IndexOf(_renderedWisps.Where(rw =>
rw.ContentPolygonShown).Select(rw =>
rw.ContentPolygon.Mesh).ToArray(), resultMesh.MeshHit) < 0 &&
528.                 Array.IndexOf(_renderedDefaultPolygons.Select(rdp =>
rdp.Mesh).ToArray(), resultMesh.MeshHit) < 0)
529.             {
530.                 resultMesh = null;
531.                 return HitTestResultBehavior.Continue;
532.             }
533.
534.             return HitTestResultBehavior.Stop;
535.         })),
536.         new PointHitTestParameters(point)
537.     );
538.
539.     return resultMesh;
540. }
541.
542. private void ShowContentPolygon(RenderedWisp renderedWisp)
543. {
544.     if (renderedWisp.ContentPolygonShown)
545.         return;
546.
547.     var cp = (renderedWisp.ContentPolygon.Parent as
ContentPolygon);
548.     var wp = (renderedWisp.WispPolygon.Parent as
WispPolygon);
549.
550.     //determining clicking group and click availability

```

```

551.         if (!string.IsNullOrEmpty(wp.ClickGroup))
552.         {
553.             if (_clickedGroups.Contains(wp.ClickGroup))
554.             {
555.                 return;
556.             }
557.             else
558.             {
559.                 _clickedGroups.Add(wp.ClickGroup);
560.             }
561.         }
562.
563.         //playing sound
564.         if (PlaySound && !string.IsNullOrEmpty(wp.ClickSound))
565.         {
566.             renderedWisp.MediaPlayer.Open(new Uri(wp.ClickSound,
UriKind.Relative));
567.             renderedWisp.MediaPlayer.Volume = 1;
568.             renderedWisp.MediaPlayer.Play();
569.         }
570.
571.         //hiding default polygons
572.         if (ReadOnly)
573.         {
574.             var defaultPolygonIds =
renderedWisp.Parent.ShowHideDefaultPolygons;
575.             var renderedDefaultPolygons =
_renderedDefaultPolygons;
576.             if (defaultPolygonIds != null)
577.             {
578.                 foreach (var id in defaultPolygonIds)
579.                 {
580.                     var rdp =
renderedDefaultPolygons.FirstOrDefault(p => p.ID == id);
581.                     if (rdp != null)
582.                     {
583.                         //increasing its hide count (0 -
visible, 1 and more - invisible)
584.                         rdp.HideCount = Math.Max(1,
rdp.HideCount + 1);
585.                         rdp.Material.Brush =
Brushes.Transparent;
586.                         if (ShowOutlines)
587.                         {
588.                             rdp.OutlinePath.Stroke =
Brushes.Transparent;
589.                             rdp.Control.Visibility =
Visibility.Hidden;
590.                         }
591.                     }
592.                 }
593.             }
594.         }
595.

```

```

596.         //backuping content polygon
597.         renderedWisp.ContentPolygonBackup = new ContentPolygon()
598.         {
599.             Content = cp.Content,
600.             ContentOrientation = cp.ContentOrientation,
601.             ContentType = cp.ContentType,
602.             CornerPoints = new PointCollection(cp.CornerPoints),
603.             FrameRate = cp.FrameRate,
604.             Loop = cp.Loop,
605.             Timeout = cp.Timeout,
606.             ZOrder = cp.ZOrder
607.         };
608.
609.         //changing wisp polygon content to click content
610.         wp.ResetClickBrush();
611.         renderedWisp.WispPolygon.Material.Brush =
wp.ClickContentBrush;
612.
613.         //updating content polygon position
614.         UpdateCornerPointPositions(cp,
renderedWisp.ContentPolygon.MatrixTransform);
615.
616.         for (int i = 0; i < 4; i++)
617.         {
618.             //preserve z order
619.             renderedWisp.ContentPolygon.Mesh.Positions[i] = new
Point3D(renderedWisp.ContentPolygon.Mesh.Positions[i].X,
renderedWisp.ContentPolygon.Mesh.Positions[i].Y, ZIncrement);
620.         }
621.
622.         //Changing it's visibility (material, outline and
controls)
623.         renderedWisp.ContentPolygon.Material.Brush =
cp.ContentBrush;
624.         if (ShowOutlines)
625.         {
626.
cp.UpdateOutline(renderedWisp.ContentPolygon.Outline);
627.             renderedWisp.ContentPolygon.OutlinePath.Stroke =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#00aff0"));
628.             var panel = renderedWisp.ContentPolygon.Control;
629.             panel.Visibility = Visibility.Visible;
630.             Canvas.SetLeft(panel, (cp.CornerPoints[1].X +
cp.CornerPoints[3].X - panel.ActualWidth) * 0.5);
631.             Canvas.SetTop(panel, Math.Max(cp.CornerPoints[1].Y,
cp.CornerPoints[3].Y));
632.         }
633.         else if (cp.Timeout > 0)
634.         {
635.             renderedWisp.HideContentPolygonTimer.Interval =
TimeSpan.FromSeconds(cp.Timeout);
636.             renderedWisp.HideContentPolygonTimer.Start();
637.         }
638.

```

```

639.         renderedWisp.ContentPolygonShown = true;
640.     }
641.
642.     private void HideContentPolygon(RenderedWisp renderedWisp)
643.     {
644.         renderedWisp.HideContentPolygonTimer.Stop();
645.
646.         if (renderedWisp.ContentPolygonShown)
647.         {
648.             var wp = (renderedWisp.WispPolygon.Parent as
other polygons
WispPolygon);
649.
650.             //determining clicking group and releasing it
651.             if (!string.IsNullOrEmpty(wp.ClickGroup))
652.             {
653.                 _clickedGroups.Remove(wp.ClickGroup);
654.             }
655.
656.             //showing default polygons which were not hide by
other polygons
657.             if (ReadOnly)
658.             {
659.                 var defaultPolygonIds =
renderedWisp.Parent.ShowHideDefaultPolygons;
660.                 var renderedDefaultPolygons =
_renderedDefaultPolygons;
661.                 if (defaultPolygonIds != null)
662.                 {
663.                     foreach (var id in defaultPolygonIds)
664.                     {
665.                         var rdp =
renderedDefaultPolygons.FirstOrDefault(p => p.ID == id);
666.                         if (rdp != null)
667.                         {
668.                             //decreasing its hide count (0 -
visible, 1 and more - invisible)
669.                             rdp.HideCount--;
670.                             if (rdp.HideCount <= 0)
671.                             {
672.                                 rdp.HideCount = 0;
673.                                 rdp.Material.Brush =
rdp.Parent.ContentBrush;
674.                                 if (ShowOutlines)
675.                                 {
676.                                     rdp.OutlinePath.Stroke =
(SolidColorBrush)(new BrushConverter().ConvertFrom("#00aff0"));
677.                                     rdp.Control.Visibility =
Visibility.Visible;
678.                                 }
679.                             }
680.                         }
681.                     }
682.                 }
683.             }

```

```

684.
685.          //changing wisp polygon content to regular content
686.          wp.ResetBrush();
687.          renderedWisp.WispPolygon.Material.Brush =
wp.ContentBrush;
688.
689.          //updating content polygon position to behind camera
position
690.          renderedWisp.ContentPolygon.Parent.ResetBrush();
691.          for (int i = 0; i < 4; i++)
692.          {
693.              renderedWisp.ContentPolygon.Mesh.Positions[i] =
new Point3D(renderedWisp.ContentPolygon.Mesh.Positions[i].X,
renderedWisp.ContentPolygon.Mesh.Positions[i].Y, 100);
694.          }
695.
696.          //hiding it (material, outline and control)
697.          renderedWisp.ContentPolygon.Material.Brush =
Brushes.Transparent;
698.          if (ShowOutlines)
699.          {
700.              renderedWisp.ContentPolygon.OutlinePath.Stroke =
Brushes.Transparent;
701.              renderedWisp.ContentPolygon.Control.Visibility =
Visibility.Hidden;
702.          }
703.
704.          renderedWisp.ContentPolygonShown = false;
705.      }
706.  }
707.
708.  private void OnNonAffineTransformControlSizeChanged(object
sender, SizeChangedEventArgs e)
709.  {
710.      if (e.PreviousSize.Width > 0)
711.      {
712.          var oldWidth = mainCamera.Width;
713.          mainCamera.Width = mainCamera.Width *
e.NewSize.Width / e.PreviousSize.Width;
714.          mainCamera.Position = new Point3D(
715.              mainCamera.Position.X + 0.5 * (mainCamera.Width
- oldWidth), // - newPosition.X + oldPosition.X,
716.              mainCamera.Position.Y + 0.5 *
(e.PreviousSize.Height - e.NewSize.Height) * mainCamera.Width /
viewport3d.ActualWidth,
717.              mainCamera.Position.Z
718.          );
719.      }
720.  }
721.
722.  private void OnNonAffineTransformControlLoaded(object
sender, RoutedEventArgs e)
723.  {

```

```

724.         overlay.Visibility = ShowOutlines ? Visibility.Visible :
Visibility.Hidden;
725.
726.         if (Visibility == System.Windows.Visibility.Visible)
727.         {
728.             LoadData();
729.             Initialize();
730.         }
731.     }
732.
733.     private void Initialize()
734.     {
735.         viewport3d.Children.Clear();
736.         overlay.Children.Clear();
737.
738.         var wisps = Wisps;
739.         var defaultPolygons = DefaultPolygons;
740.         if (wisps == null)
741.             return;
742.
743.         var modelVisual3D = new ModelVisual3D();
744.         var model3DGroup = new Model3DGroup();
745.         //for each provided brush one plane is created, each
next plane is a little further (first on is the most visible)
746.
747.         var models = new List<GeometryModel3D>();
748.
749.         var wispsCount = wisps.Length;
750.         var count = defaultPolygons != null ?
defaultPolygons.Length : 0;
751.
752.         #region Default polygons initialization
753.         _renderedDefaultPolygons = new
RenderedDefaultPolygon[count];
754.         for (int i = 0; i < count; i++)
755.         {
756.             double z = i < count ?
757.                 ((defaultPolygons[i].ZOrder > 0 ?
defaultPolygons[i].ZOrder : i) + wispsCount + 1) * ZIncrement :
ZIncrement;
758.             //double z = (zOrder != null && zOrder.Length > i) ?
zOrder[i] * ZIncrement : i * ZIncrement;
759.
760.             _renderedDefaultPolygons[i] = new
RenderedDefaultPolygon()
761.             {
762.                 Parent = defaultPolygons[i],
763.                 PointsTransformed = new Point3D[4],
764.                 ID = defaultPolygons[i].ID
765.             };
766.
767.             var pointsTransformed = new Point3D[] {
768.                 new Point3D(0, 0, z),
769.                 new Point3D(0, 1, z),
770.

```

```

771.                new Point3D(1, 0, z),
772.                new Point3D(1, 1, z)
773.            };
774.
775.            _renderedDefaultPolygons[i].Mesh = new
MeshGeometry3D()
776.            {
777.                //initial positions of planes
778.                Positions = new
Point3DCollection(pointsTransformed),
779.                TextureCoordinates = new PointCollection(new
Point[] {
780.                    new Point(0, 0),
781.                    new Point(0, 1),
782.                    new Point(1, 0),
783.                    new Point(1, 1)
784.                }),
785.                TriangleIndices = new Int32Collection(new
Int32[] {
786.                    0, 2, 1, 2, 3, 1
787.                })
788.            };
789.            _renderedDefaultPolygons[i].MatrixTransform = new
MatrixTransform3D();
790.            if (ShowContent)
791.            {
792.                //creating brush
793.                _renderedDefaultPolygons[i].Material = new
DiffuseMaterial(defaultPolygons[i].ContentBrush);
794.            }
795.            else
796.            {
797.                _renderedDefaultPolygons[i].Material = new
DiffuseMaterial(Brushes.Transparent);
798.            }
799.
800.            if (ShowOutlines)
801.            {
802.                //creating outline
803.                _renderedDefaultPolygons[i].Outline =
defaultPolygons[i].CreateOutline();
804.
805.                //adding to outlines collections
806.                overlay.Children.Add(new Path()
807.                {
808.                    Stroke = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#00aff0")),
809.                    StrokeThickness = 2,
810.                    Data = new PathGeometry()
811.                    {
812.                        Figures = new PathFigureCollection(new
PathFigure[] {
813.                            _renderedDefaultPolygons[i].Outline

```

```

814.                                     ))
815.                                 }
816.                             });
817.
818.                             var name = new TextBlock()
819.                             {
820.                                 Text = defaultPolygons[i].ID,
821.                                 TextAlignment = TextAlignment.Center,
822.                                 Foreground = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#00aff0")),
823.                             };
824.                             _renderedDefaultPolygons[i].Control = name;
825.
826.                             overlay.Children.Add(name);
827.                             var polygon = defaultPolygons[i];
828.                             name.Loaded += new RoutedEventHandler((s, args)
=>
829.                             {
830.                                 Canvas.SetLeft(name,
(polygon.CornerPoints[1].X + polygon.CornerPoints[3].X -
name.ActualWidth) * 0.5);
831.                                 Canvas.SetTop(name,
Math.Max(polygon.CornerPoints[1].Y, polygon.CornerPoints[3].Y));
832.                             });
833.                             }
834.                             models.Add(
835.                                 new GeometryModel3D()
836.                                 {
837.                                     Geometry = _renderedDefaultPolygons[i].Mesh,
838.                                     Material =
_renderedDefaultPolygons[i].Material,
839.                                     Transform =
_renderedDefaultPolygons[i].MatrixTransform
840.                                 }
841.                             );
842.                             }
843.                             #endregion
844.
845.                             #region Wisps initialization
846.                             _renderedWisps = new RenderedWisp[wispsCount];
847.                             for (int i = 0; i < wispsCount; i++)
848.                             {
849.                                 double z = i < wispsCount ?
850.                                     (((wisps[i].WispPolygon.ZOrder > 0) ?
wisps[i].WispPolygon.ZOrder : i) + 1) * ZIncrement :
851.                                     ZIncrement;
852.                                 //double z = (zOrder != null && zOrder.Length > i) ?
zOrder[i] * ZIncrement : i * ZIncrement;
853.
854.                                 var timer = new DispatcherTimer();
855.                                 timer.Tick += OnHideContentPolygonTimerTick;
856.
857.                                 _renderedWisps[i] = new RenderedWisp()
858.                                 {

```

```

859.         HideContentPolygonTimer = timer,
860.         MediaPlayer = new MediaPlayer(),
861.         Parent = wisps[i],
862.         WispPolygon = new RenderedPolygon()
863.         {
864.             Parent = wisps[i].WispPolygon,
865.             PointsTransformed = new Point3D[4]
866.         },
867.         ContentPolygon = new RenderedPolygon()
868.         {
869.             Parent = wisps[i].ContentPolygon,
870.             PointsTransformed = new Point3D[4]
871.         }
872.     };
873.
874.     timer.Tag = _renderedWisps[i];
875.
876.     var pointsTransformed = new Point3D[] {
877.         new Point3D(0, 0, z),
878.         new Point3D(0, 1, z),
879.         new Point3D(1, 0, z),
880.         new Point3D(1, 1, z)
881.     };
882.
883.     _renderedWisps[i].WispPolygon.Mesh = new
MeshGeometry3D()
884.     {
885.         //initial positions of planes
886.         Positions = new
Point3DCollection(pointsTransformed),
887.         TextureCoordinates = new PointCollection(new
Point[] {
888.             new Point(0, 0),
889.             new Point(0, 1),
890.             new Point(1, 0),
891.             new Point(1, 1)
892.         }),
893.         TriangleIndices = new Int32Collection(new
Int32[] {
894.             0, 2, 1, 2, 3, 1
895.         })
896.     };
897.     _renderedWisps[i].WispPolygon.MatrixTransform = new
MatrixTransform3D();
898.     if (ShowContent)
899.     {
900.         //creating brush
901.         _renderedWisps[i].WispPolygon.Material = new
DiffuseMaterial(wisps[i].WispPolygon.ContentBrush);
902.     }
903.     else
904.     {
905.         _renderedWisps[i].WispPolygon.Material = new
DiffuseMaterial(Brushes.Transparent);

```

```

906.         }
907.
908.         if (ShowOutlines)
909.         {
910.             //creating outline
911.             _renderedWisps[i].WispPolygon.Outline =
wisps[i].WispPolygon.CreateOutline();
912.
913.             //adding to outlines collections
914.             overlay.Children.Add(new Path()
915.             {
916.                 Stroke = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#00aff0")),
917.                 StrokeThickness = 2,
918.                 Data = new PathGeometry()
919.                 {
920.                     Figures = new PathFigureCollection(new
PathFigure[] {
921.                         _renderedWisps[i].WispPolygon.Outline
922.                             })
923.                     }
924.                 });
925.
926.             var name = new TextBlock()
927.             {
928.                 Text = wisps[i].Name,
929.                 TextAlignment = TextAlignment.Center,
930.                 Foreground = (SolidColorBrush)(new
BrushConverter().ConvertFrom("#00aff0")),
931.             };
932.             _renderedWisps[i].WispPolygon.Control = name;
933.
934.             overlay.Children.Add(name);
935.             var polygon = wisps[i].WispPolygon;
936.             name.Loaded += new RoutedEventHandler((s, args)
=>
937.             {
938.                 Canvas.SetLeft(name,
(polygon.CornerPoints[1].X + polygon.CornerPoints[3].X -
name.ActualWidth) * 0.5);
939.                 Canvas.SetTop(name,
Math.Max(polygon.CornerPoints[1].Y, polygon.CornerPoints[3].Y));
940.             });
941.         }
942.         models.Add(
943.             new GeometryModel3D()
944.             {
945.                 Geometry =
_renderedWisps[i].WispPolygon.Mesh,
946.                 Material =
_renderedWisps[i].WispPolygon.Material,
947.                 Transform =
_renderedWisps[i].WispPolygon.MatrixTransform

```

```

948.         }
949.     );
950.
951.     _renderedWisps[i].ContentPolygon.Mesh = new
MeshGeometry3D()
952.     {
953.         //initial pozitions of planes
954.         Positions = new
Point3DCollection(pointsTransformed),
955.         TextureCoordinates = new PointCollection(new
Point[] {
956.             new Point(0, 0),
957.             new Point(0, 1),
958.             new Point(1, 0),
959.             new Point(1, 1)
960.         }),
961.         TriangleIndices = new Int32Collection(new
Int32[] {
962.             0, 2, 1, 2, 3, 1
963.         })
964.     };
965.     _renderedWisps[i].ContentPolygon.MatrixTransform =
new MatrixTransform3D();
966.
967.     _renderedWisps[i].ContentPolygon.Material = new
DiffuseMaterial(Brushes.Transparent);
968.
969.     if (ShowOutlines)
970.     {
971.         _renderedWisps[i].ContentPolygon.Outline =
Polygon.CreateEmptyOutline();
972.         _renderedWisps[i].ContentPolygon.OutlinePath =
new Path()
973.         {
974.             Stroke = Brushes.Transparent,
975.             StrokeThickness = 2,
976.             Data = new PathGeometry()
977.             {
978.                 Figures = new PathFigureCollection(new
PathFigure[] {
979.                     _renderedWisps[i].ContentPolygon.Outline
980.                 })
981.             }
982.         };
983.
overlay.Children.Add(_renderedWisps[i].ContentPolygon.OutlinePath);
984.
985.         //container for controls
986.         var panel = new StackPanel()
987.         {
988.             Visibility = Visibility.Hidden,
989.             Orientation = Orientation.Horizontal
990.         };

```

```

991.
992.         //Save Content button
993.         var button = new Button()
994.         {
995.             Content = "Save Content",
996.             Margin = new Thickness(10)
997.         };
998.         button.Click += OnSaveContentButtonClick;
999.         button.Tag = _renderedWisps[i];
1000.        panel.Children.Add(button);
1001.
1002.        //Rotate image
1003.        var image = new Image()
1004.        {
1005.            Source = new BitmapImage(new
1006.                Uri("/rotate.png", UriKind.Relative)),
1007.            Margin = new Thickness(0, 10, 0, 10),
1008.            Width = 20,
1009.            Height = 20
1010.        };
1011.        image.MouseUp += OnRotateContentButtonClick;
1012.        image.Tag = _renderedWisps[i];
1013.        //nice antializings
1014.        RenderOptions.SetBitmapScalingMode(image,
1015.            BitmapScalingMode.Fant);
1016.        panel.Children.Add(image);
1017.        overlay.Children.Add(panel);
1018.        _renderedWisps[i].ContentPolygon.Control =
1019.            panel;
1020.        }
1021.
1022.        models.Add(
1023.            new GeometryModel3D()
1024.            {
1025.                Geometry =
1026.                    _renderedWisps[i].ContentPolygon.Mesh,
1027.                Material =
1028.                    _renderedWisps[i].ContentPolygon.Material,
1029.                Transform =
1030.                    _renderedWisps[i].ContentPolygon.MatrixTransform
1031.            }
1032.        );
1033.    }
1034.    #endregion
1035.
1036.    _loaded = true;
1037.
1038.    UpdateCornerPointPositions();
1039.    //storing models in correct z-order
1040.    models.Sort(new Comparison<GeometryModel3D>((a, b) =>
1041.        a.Geometry.Bounds.Z.CompareTo(b.Geometry.Bounds.Z)));
1042.    models.ForEach(o => model3DGroup.Children.Add(o));
1043.    model3DGroup.Children.Add(new
1044.        AmbientLight(Colors.White));

```

```

1037.         modelVisual3D.Content = model3DGroup;
1038.         this.viewport3d.Children.Add(modelVisual3D);
1039.     }
1040.
1041.     private void OnSaveContentButtonClick(object sender,
RoutedEventArgs e)
1042.     {
1043.         var fe = (sender as FrameworkElement);
1044.         if (fe == null)
1045.             return;
1046.
1047.         var rw = fe.Tag as RenderedWisp;
1048.         if (rw == null)
1049.             return;
1050.
1051.         SaveData();
1052.         HideContentPolygon(rw);
1053.     }
1054.
1055.     /// <summary>
1056.     /// Loads data from provided xml file.
1057.     /// </summary>
1058.     private void LoadData()
1059.     {
1060.         if (!string.IsNullOrEmpty(StorageFile) &&
System.IO.File.Exists(StorageFile))
1061.         {
1062.             try
1063.             {
1064.                 using (var fs =
System.IO.File.OpenRead(StorageFile))
1065.                 {
1066.                     XmlSerializer xs = new
XmlSerializer(typeof(ObjectStorage), new Type[] {
1067.                         typeof(Wisp[]),
1068.                         typeof(Wisp),
1069.                         typeof(ContentPolygon),
1070.                         typeof(WispPolygon),
1071.                         typeof(Polygon),
1072.                         typeof(DefaultPolygon[]),
1073.                         typeof(DefaultPolygon)
1074.                     });
1075.                     var objectStorage =
(ObjectStorage)xs.Deserialize(fs);
1076.                     _internalChange = true;
1077.                     Wisps = objectStorage.Wisps;
1078.                     DefaultPolygons =
objectStorage.DefaultPolygons;
1079.                     _internalChange = false;
1080.                 }
1081.             }
1082.             catch { }
1083.         }
1084.     }

```

```

1085.
1086.     /// <summary>
1087.     /// Saves data to provided xml file.
1088.     /// </summary>
1089.     private void SaveData()
1090.     {
1091.         if (!string.IsNullOrEmpty(StorageFile) && Wisps != null)
1092.         {
1093.             using (var fs = System.IO.File.Create(StorageFile))
1094.             {
1095.                 XmlSerializer xs = new
1096.                 XmlSerializer(typeof(ObjectStorage), new Type[] {
1097.                     typeof(Wisp[]),
1098.                     typeof(Wisp),
1099.                     typeof(ContentPolygon),
1100.                     typeof(WispPolygon),
1101.                     typeof(Polygon),
1102.                     typeof(DefaultPolygon[]),
1103.                     typeof(DefaultPolygon)
1104.                 });
1105.                 var objectStorage = new ObjectStorage()
1106.                 {
1107.                     Wisps = this.Wisps,
1108.                     DefaultPolygons = this.DefaultPolygons
1109.                 };
1110.                 xs.Serialize(fs, objectStorage);
1111.                 fs.Close();
1112.             }
1113.         }
1114.     }
1115.
1116.     /// <summary>
1117.     /// Rotates content of Content polygon
1118.     /// </summary>
1119.     /// <param name="sender"></param>
1120.     /// <param name="e"></param>
1121.     private void OnRotateContentButtonClick(object sender,
1122.         MouseButtonEventArgs e)
1123.     {
1124.         var fe = (sender as FrameworkElement);
1125.         if (fe == null)
1126.             return;
1127.
1128.         var rw = fe.Tag as RenderedWisp;
1129.         if (rw == null)
1130.             return;
1131.
1132.         rw.Parent.ContentPolygon.ContentOrientation -= 90;
1133.         if (rw.Parent.ContentPolygon.ContentOrientation < 0)
1134.         {
1135.             rw.Parent.ContentPolygon.ContentOrientation += 360;
1136.         }

```

```

1137.         rw.ContentPolygon.Material.Brush =
1138.         rw.ContentPolygon.Parent.ContentBrush;
1139.     }
1140.     /// <summary>
1141.     /// Updates transformations matrices according new corner
1142.     points positions
1143.     /// </summary>
1144.     private void UpdateCornerPointPositions()
1145.     {
1146.         if (!_loaded)
1147.             return;
1148.
1149.         //var count = Brushes.Length;
1150.         var wisps = Wisps;
1151.         var defaultPolygons = DefaultPolygons;
1152.
1153.         if (wisps == null)
1154.             return;
1155.
1156.         var count = wisps.Length;
1157.         //for each provided brush one plane is created, each
1158.         next plane is a little further (first on is the most visible)
1159.         for (int i = 0; i < count; i++)
1160.         {
1161.             UpdateCornerPointPositions(wisps[i].WispPolygon,
1162.             _renderedWisps[i].WispPolygon.MatrixTransform, i * ZIncrement);
1163.         }
1164.
1165.         if (defaultPolygons != null)
1166.         {
1167.             var dpCount = defaultPolygons.Length;
1168.             for (int i = 0; i < dpCount; i++)
1169.             {
1170.                 UpdateCornerPointPositions(defaultPolygons[i],
1171.                 _renderedDefaultPolygons[i].MatrixTransform, (count + i) *
1172.                 ZIncrement);
1173.             }
1174.         }
1175.     }
1176.
1177.     private void UpdateCornerPointPositions(Polygon polygon,
1178.     MatrixTransform3D matrix, double defaultZ = ZIncrement)
1179.     {
1180.         double z = (polygon.ZOrder > 0) ? polygon.ZOrder *
1181.         ZIncrement : defaultZ;
1182.
1183.         var pointsTransformed = new Point3D[] {
1184.             new Point3D(0, 0, z),
1185.             new Point3D(0, 1, z),
1186.             new Point3D(1, 0, z),
1187.             new Point3D(1, 1, z)
1188.         };
1189.         for (int j = 0; j < 4; j++)

```

```

1183.      {
1184.          //transforming from display to 3d coordinates or
using default values
1185.          var inputCornerPointRelative = (polygon.CornerPoints
!= null && polygon.CornerPoints.Count > j) ?
1186.              Simple2Dto3D(viewport3d,
polygon.CornerPoints[j]) : DefaultCornerPoints[j];
1187.          for (int k = 0; k < 4; k++)
1188.          {
1189.              //it's needed to prevent NaN in matrix
calculations (two points cannot have the same coordinates)
1190.              if (j != k && Math.Abs(pointsTransformed[k].X -
inputCornerPointRelative.X) < 0.0001
1191.                  && Math.Abs(pointsTransformed[k].Y -
inputCornerPointRelative.Y) < 0.0001)
1192.              {
1193.                  inputCornerPointRelative.X =
pointsTransformed[k].X + 0.0001;
1194.                  inputCornerPointRelative.Y =
pointsTransformed[k].Y + 0.0001;
1195.              }
1196.          }
1197.
1198.          pointsTransformed[j].X = inputCornerPointRelative.X;
1199.          pointsTransformed[j].Y = inputCornerPointRelative.Y;
1200.      }
1201.      //updating matrix
1202.      matrix.Matrix =
CalculateNonAffineTransform(pointsTransformed);
1203.      //updating corner point positons (in case of incomplete
set provided)
1204.      var outputCornerPoints = new Point[4];
1205.      for (int j = 0; j < 4; j++)
1206.      {
1207.          outputCornerPoints[j] = Simple3Dto2D(viewport3d,
pointsTransformed[j]); // new Point(_pointsTransformed[i][j].X,
_pointsTransformed[i][j].Y);
1208.      }
1209.
1210.      _internalChange = true;
1211.      polygon.CornerPoints = new
PointCollection(outputCornerPoints);
1212.      _internalChange = false;
1213.  }
1214.
1215.      // The input array of points describes a 2D rectangle
1216.      // (with Z assumed to be zero) in the order
1217.      // lower-left, upper-left, lower-right, upper-right.
1218.      // The returned transform maps the points (0, 0, 0),
1219.      // (0, 1, 0), (1, 0, 0), and (1, 1, 0) to these points.
1220.      private Matrix3D CalculateNonAffineTransform(Point3D[]
points)
1221.      {
1222.          // Affine transform

```

```

1223.          // -----
1224.          // This matrix maps (0, 0) --> (x0, y0)
1225.          //                   (0, 1) --> (x1, y1)
1226.          //                   (1, 0) --> (x2, y2)
1227.          //                   (1, 1) --> (x2 + x1 + x0, y2 + y1 +
y0)
1228.          Matrix3D A = new Matrix3D();
1229.          A.M11 = points[2].X - points[0].X;
1230.          A.M12 = points[2].Y - points[0].Y;
1231.          A.M21 = points[1].X - points[0].X;
1232.          A.M22 = points[1].Y - points[0].Y;
1233.          A.OffsetX = points[0].X;
1234.          A.OffsetY = points[0].Y;
1235.
1236.          // Calculate point (a, b) that get mapped by the affine
transform to (x3, y3)
1237.          double den = A.M11 * A.M22 - A.M12 * A.M21;
1238.          double a = (A.M22 * points[3].X - A.M21 * points[3].Y +
1239.                      A.M21 * A.OffsetY - A.M22 * A.OffsetX) /
den;
1240.
1241.          double b = (A.M11 * points[3].Y - A.M12 * points[3].X +
1242.                      A.M12 * A.OffsetX - A.M11 * A.OffsetY) /
den;
1243.
1244.          // Non-affine transform
1245.          // -----
1246.          // This matrix maps (0, 0) --> (0, 0)
1247.          //                   (0, 1) --> (0, 1)
1248.          //                   (1, 0) --> (1, 0)
1249.          //                   (1, 1) --> (a, b)
1250.
1251.          Matrix3D B = new Matrix3D();
1252.          B.M11 = a / (a + b - 1);
1253.          B.M22 = b / (a + b - 1);
1254.          B.M14 = B.M11 - 1;
1255.          B.M24 = B.M22 - 1;
1256.
1257.          return B * A;
1258.      }
1259.
1260.          // The following two methods only work with
OrthographicCamera,
1261.          // with LookDirection of (0, 0, -1) and UpDirection of (0,
1, 0).
1262.          // More advanced conversion routines can be found in the
1263.          // Petzold.Media3D library.
1264.
1265.          // Converts a 2D point in device-independent coordinates
relative
1266.          // to Viewport3D to 3D space.
1267.          private Point3D Simple2Dto3D(Viewport3D vp, Point pt, double
z = 0.0)
1268.          {

```

```

1269.         OrthographicCamera cam = CheckRestrictions(vp);
1270.         double scale = cam.Width / vp.ActualWidth;
1271.         double x = scale * (pt.X - vp.ActualWidth / 2) +
cam.Position.X;
1272.         double y = scale * (vp.ActualHeight / 2 - pt.Y) +
cam.Position.Y;
1273.
1274.         return new Point3D(x, y, z);
1275.     }
1276.
1277.     // Converts a 3D point to 2D in device-independent
coordinates
1278.     // relative to Viewport3D.
1279.     private Point Simple3Dto2D(Viewport3D vp, Point3D point)
1280.     {
1281.         OrthographicCamera cam = CheckRestrictions(vp);
1282.         double scale = vp.ActualWidth / cam.Width;
1283.         double x = vp.ActualWidth / 2 + scale * (point.X -
cam.Position.X);
1284.         double y = vp.ActualHeight / 2 - scale * (point.Y -
cam.Position.Y);
1285.         return new Point(x, y);
1286.     }
1287.
1288.     private OrthographicCamera CheckRestrictions(Viewport3D vp)
1289.     {
1290.         OrthographicCamera cam = vp.Camera as
OrthographicCamera;
1291.
1292.         if (cam == null)
1293.             throw new ArgumentException("Camera must be
OrthographicCamera");
1294.
1295.         if (cam.LookDirection != new Vector3D(0, 0, -1))
1296.             throw new ArgumentException("Camera LookDirection
must be (0, 0, -1)");
1297.
1298.         if (cam.UpDirection != new Vector3D(0, 1, 0))
1299.             throw new ArgumentException("Camera UpDirection must
be (0, 1, 0)");
1300.
1301.         return cam;
1302.     }
1303.
1304.     private RenderedWisp[] _renderedWisps;
1305.     private RenderedDefaultPolygon[] _renderedDefaultPolygons;
1306.
1307.     /// <summary>
1308.     /// Indicates if user drags mouse
1309.     /// </summary>
1310.     private bool _isDragging;
1311.
1312.     private bool _isDraggingContent;
1313.

```

```

1314.         private RenderedWisp _draggingWisp = null;
1315.         private RenderedPolygon _draggingPolygon = null;
1316.
1317.         /// <summary>
1318.         /// Indicates index of dragging vertex
1319.         /// </summary>
1320.         private int _draggingVertexIndex;
1321.
1322.         /// <summary>
1323.         /// Start position for dragging mesh
1324.         /// </summary>
1325.         private Point3D[] _startDraggingMeshPoints;
1326.
1327.         /// <summary>
1328.         /// Start position for mouse while dragging a mesh
1329.         /// </summary>
1330.         private Point3D _startDraggingPoint;
1331.
1332.         /// <summary>
1333.         /// Start position of mouse while dragging
1334.         /// </summary>
1335.         private Point _startDraggingMousePoint;
1336.
1337.         /// <summary>
1338.         /// Indicates that Control Points assigned from within
control class
1339.         /// </summary>
1340.         private bool _internalChange = false;
1341.
1342.         /// <summary>
1343.         /// Indicates that control is loaded
1344.         /// </summary>
1345.         private bool _loaded = false;
1346.
1347.         /// <summary>
1348.         /// Stores a list of all clicked click groups
1349.         /// </summary>
1350.         private List<string> _clickedGroups = new List<string>();
1351.     }
1352. }
1353.

```

CLAIMS:

1. A system for projecting interactive images on a three-dimensional object comprising:
a camera capable of providing distance-to-target information;
a projector and the camera both positioned adjacent the object, such that the object is positioned within a field of view of the projector;
a computer, connected to the camera and the projector, programmed to carry out a computer implemented method having the steps of:
scanning the object within the field of view with the camera;
projecting a trigger area on a surface of the object;
warping a polygon to match the surface;
associating the polygon with the trigger area;
associating a set of content with the polygon;
warping the set of content to the polygon;
determining a touch event; and,
projecting the warped set of content on the surface.
2. The system of claim 1 wherein the step of scanning further comprises:
storing a baseline image of the surface.
3. The system of claim 1 wherein the computer implemented method further comprises the steps of:
calibrating the trigger area; and,
saving the calibrated trigger area.
4. The system of claim 1 wherein the computer implemented method comprises the further

step of:

determining a touch event.

5. The system of claim 1 wherein the computer implemented method comprises the further

step of:

determining the presence of an end event.

6. The system of claim 1 wherein the computer implemented method comprises the further

step of:

determining the presence of a user within the field of view.

7. The system of claim 1 wherein the computer implemented method comprises the further

step of:

determining the identity of a user within the field of view; and,

projecting a user specific warped set of content on the surface.

8. The system of claim 1 further comprising:

a mirror positioned adjacent the projector such that the field of view is redirected by the mirror.

9. The system of claim 1 further comprising:

an arm connected to and supporting the camera, the projector, and the computer.

10. The system of claim 1 further comprising:

a floor stand connected to and supporting the camera, the projector, and the computer.

11. The system of claim 1 further comprising:

an arm connected to and supporting the camera, the projector, and the computer;
a floor stand connected to a placard, wherein the placard is within the field of view; and,
the computer implemented method comprises the further step of:

projecting the warped set of content on the placard.

12. The system of claim 1 further comprising:

a secondary projector and a secondary camera both positioned adjacent the object, such that the object is positioned within a secondary field of view of the secondary projector;

a first side of the object within the field of view and a secondary side of the object within the secondary field of view;

the computer implemented method comprises the further step of:

projecting the warped set of content on the first side; and,

a secondary computer connected to the secondary camera and the secondary projector, programmed to carry out a secondary computer implemented method having the step of:

projecting a secondary warped set of content on the secondary side.

13. A method for projecting an interactive image on a three-dimensional object comprising the steps of:

providing a camera, capable of providing distance-to-target information, and a projector connected to a computer;

positioning the camera and the projector adjacent the object, such that the object is positioned within a field of view of the projector;

defining the positions of a set of trigger areas on a surface of the object;

projecting the set of trigger areas on the surface;

determining the presence of a touch event;

repeating the steps of projecting the set of trigger areas and determining the presence of the touch event until the touch event is detected;

projecting a set of content on the surface;

determining the presence of an end event;

repeating the steps of projecting the set of content and determining the presence of an end event until the end event has occurred; and,

terminating the projecting.

14. The method of claim 13 wherein the step of defining the positions of the set of trigger areas further comprises the steps of:

scanning the object within the field of view with the camera;

storing a baseline image of the surface;

calibrating the set of trigger areas; and,

saving the calibrated set of trigger areas.

15. The method of claim 13 wherein the step of projecting the set of content on the surface further comprises the steps of:

warping a polygon to match the surface;

associating the polygon with the trigger area;

associating a set of content with the polygon; and,

warping the set of content to the polygon.

16. The method of claim 13 further comprising the steps of:

performing a facial recognition process of a user within the field of view;

determining the identity of the user; and,

projecting a set of user specific content on the surface.

17. The method of claim 16 wherein the step of performing the facial recognition process further comprises the steps of:

recording facial features of the user with the camera;

measuring a set of distances between recorded facial features;

creating a facial map; and,

comparing the facial map to a set of prerecorded digital images of the user.

18. The method of claim 13 further comprising the step of:

providing an arm connected to and supporting the camera, the projector, and the computer;

providing a floor stand connected to a placard, wherein the placard is positioned within the field of view;

projecting the set of content on the placard.

19. The method of claim 13 further comprising the step of:

providing a second camera and a second projector both connected to a second computer;

positioning the second camera and the second projector adjacent the object, such that the object is positioned within a second field of view of the second projector;

providing a first side of the object within the field of view and a second side of the object within the second field of view;

projecting the set of content from the projector on the first side; and,

projecting a second set of content from the second projector on the second side.

20. A system for projecting an interactive image onto an object comprising:

an infrared sensor;

a projector;

a controller operatively connected to the infrared sensor and the projector;

the controller programmed to:

transform a content image to conform to a surface of the object resulting in a mapped content image;

transform a trigger image to conform to the surface of the object resulting in a mapped trigger image;

send a mapped trigger image signal to the projector whereby the projector projects the mapped trigger image onto the surface of the object;

receive a data signal from the infrared sensor;

generate a trigger signal based on the data signal; and,

in response to the trigger signal, send a mapped content image signal to the projector, whereby the projector projects the mapped content image onto the surface of the object.

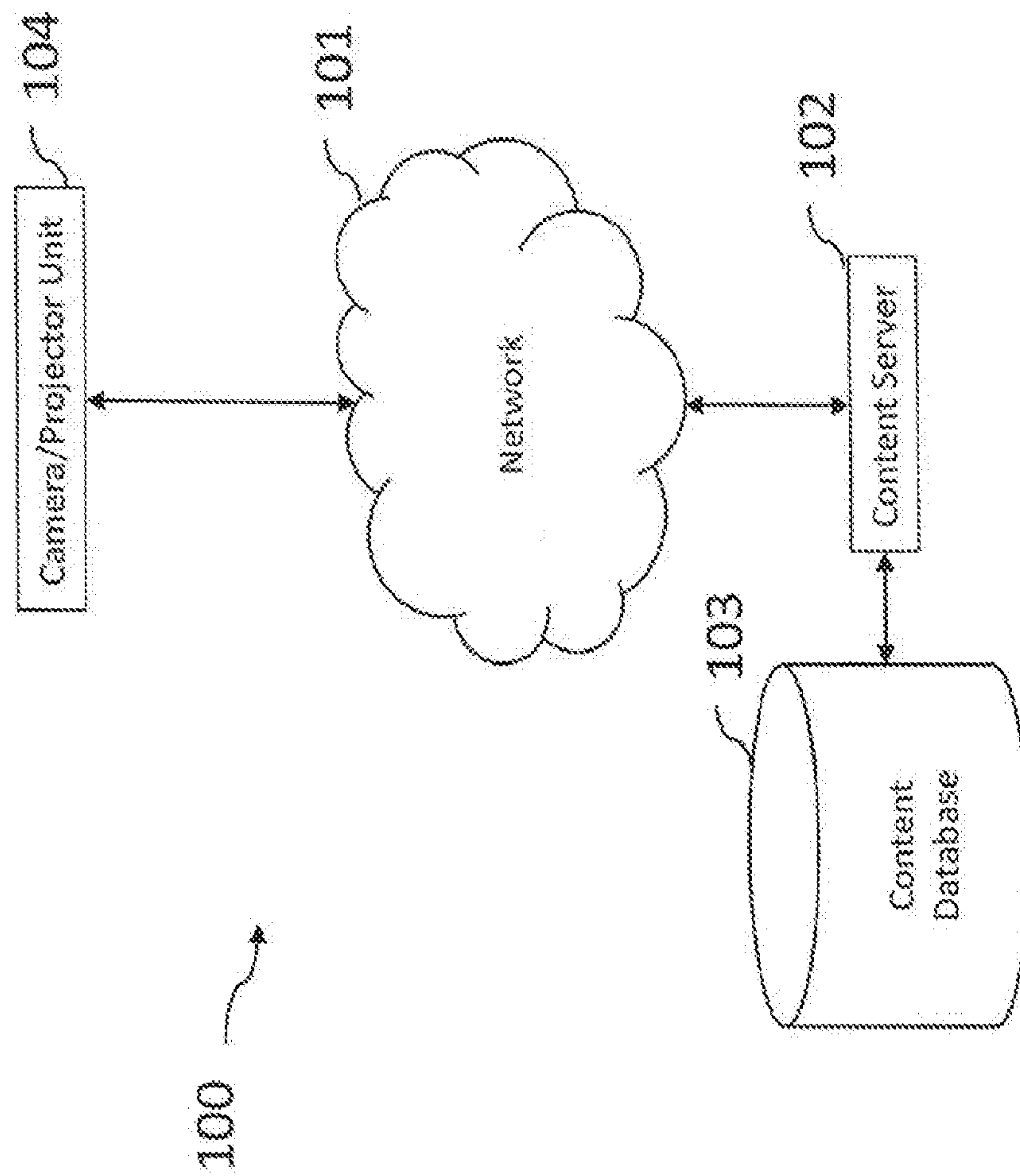


FIG. 1

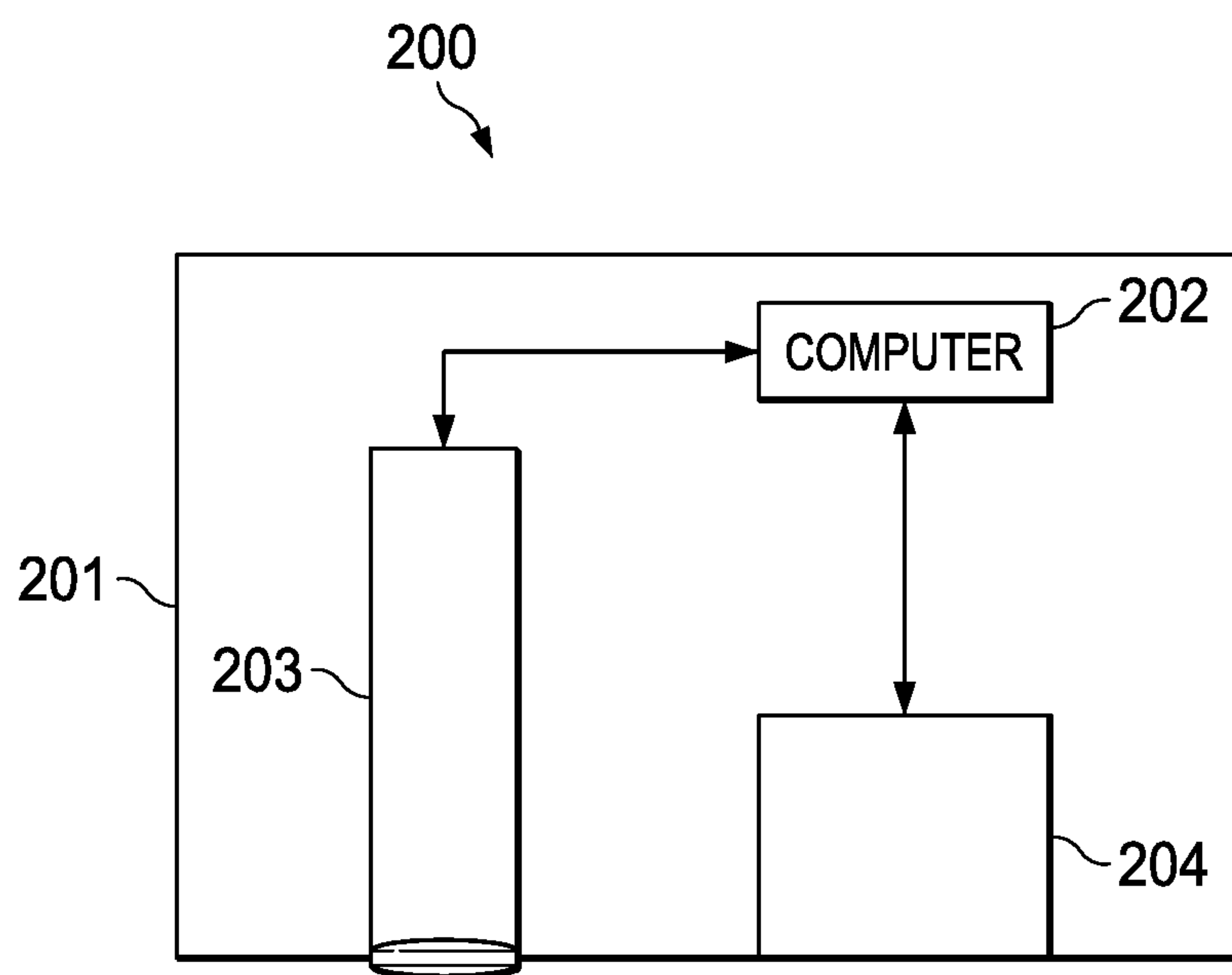


FIG. 2A

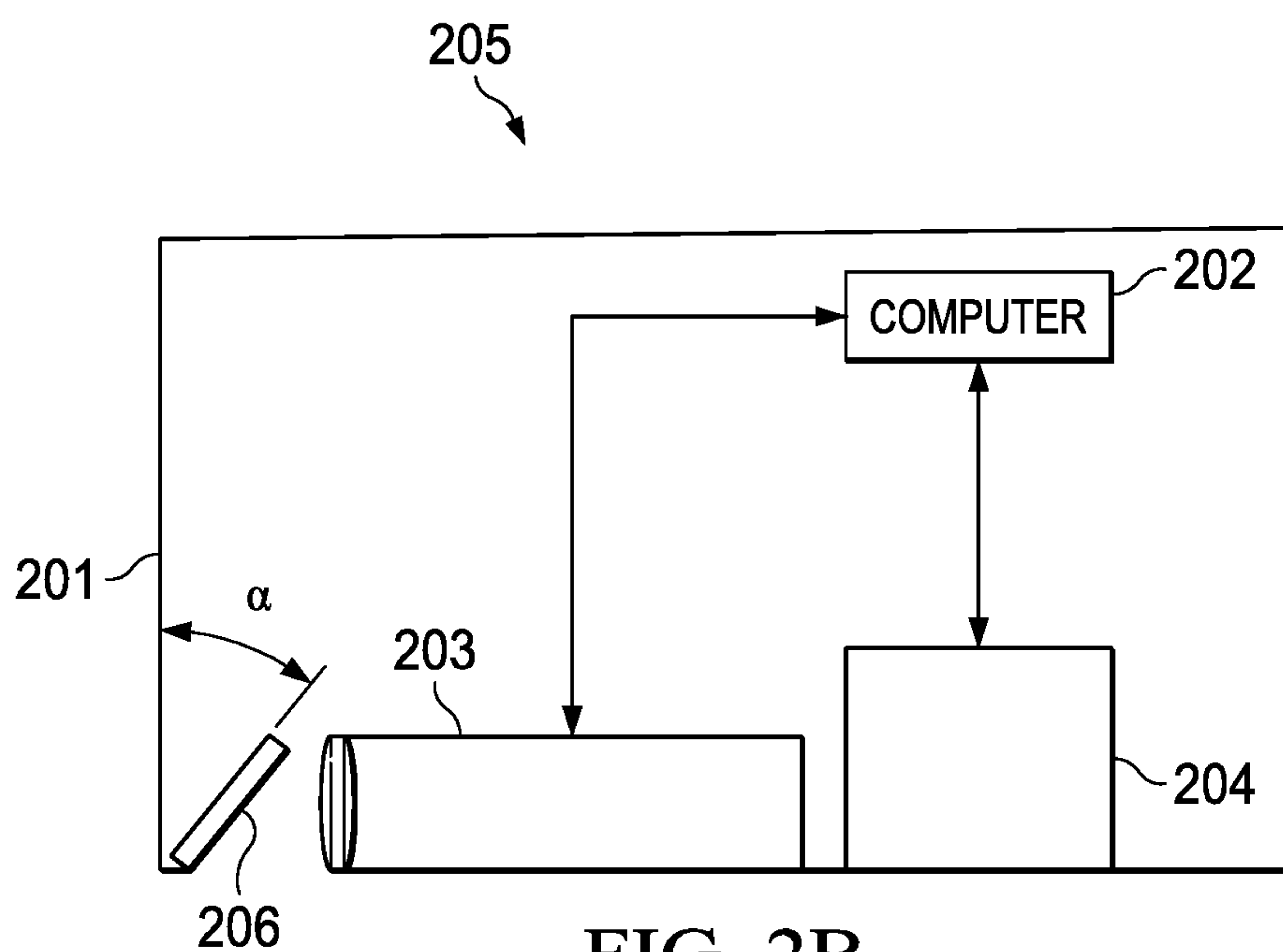


FIG. 2B

3/26

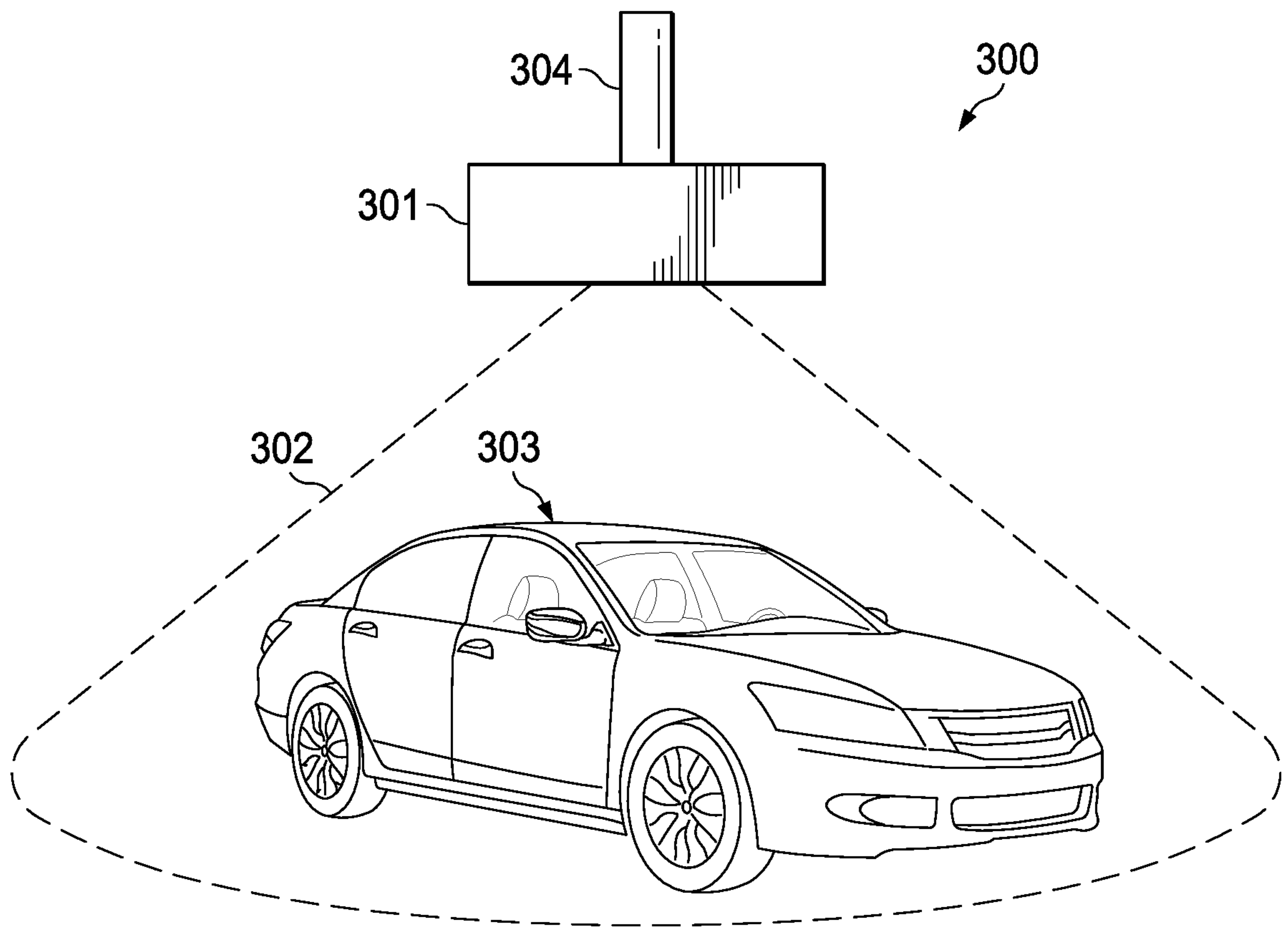


FIG. 3

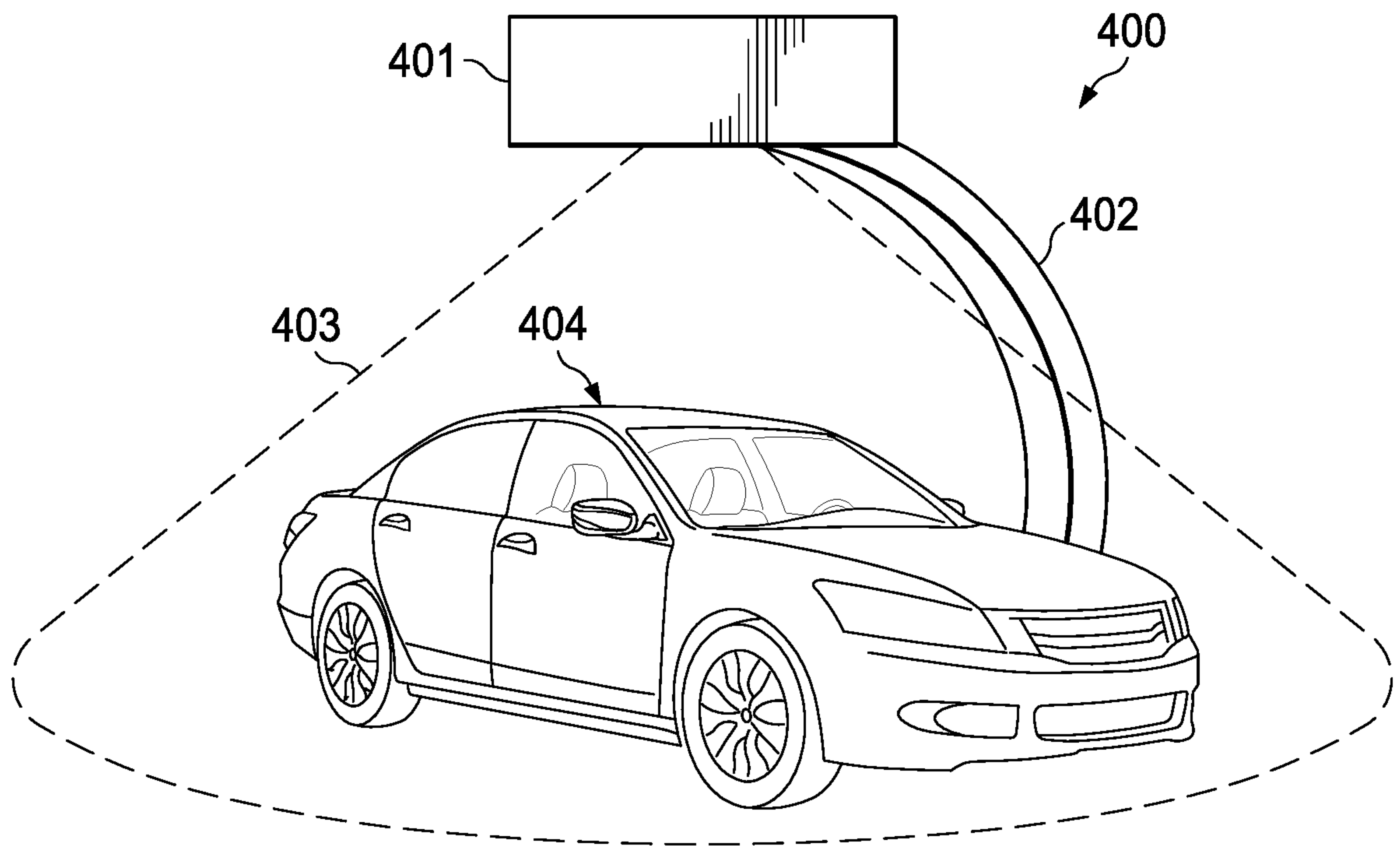


FIG. 4

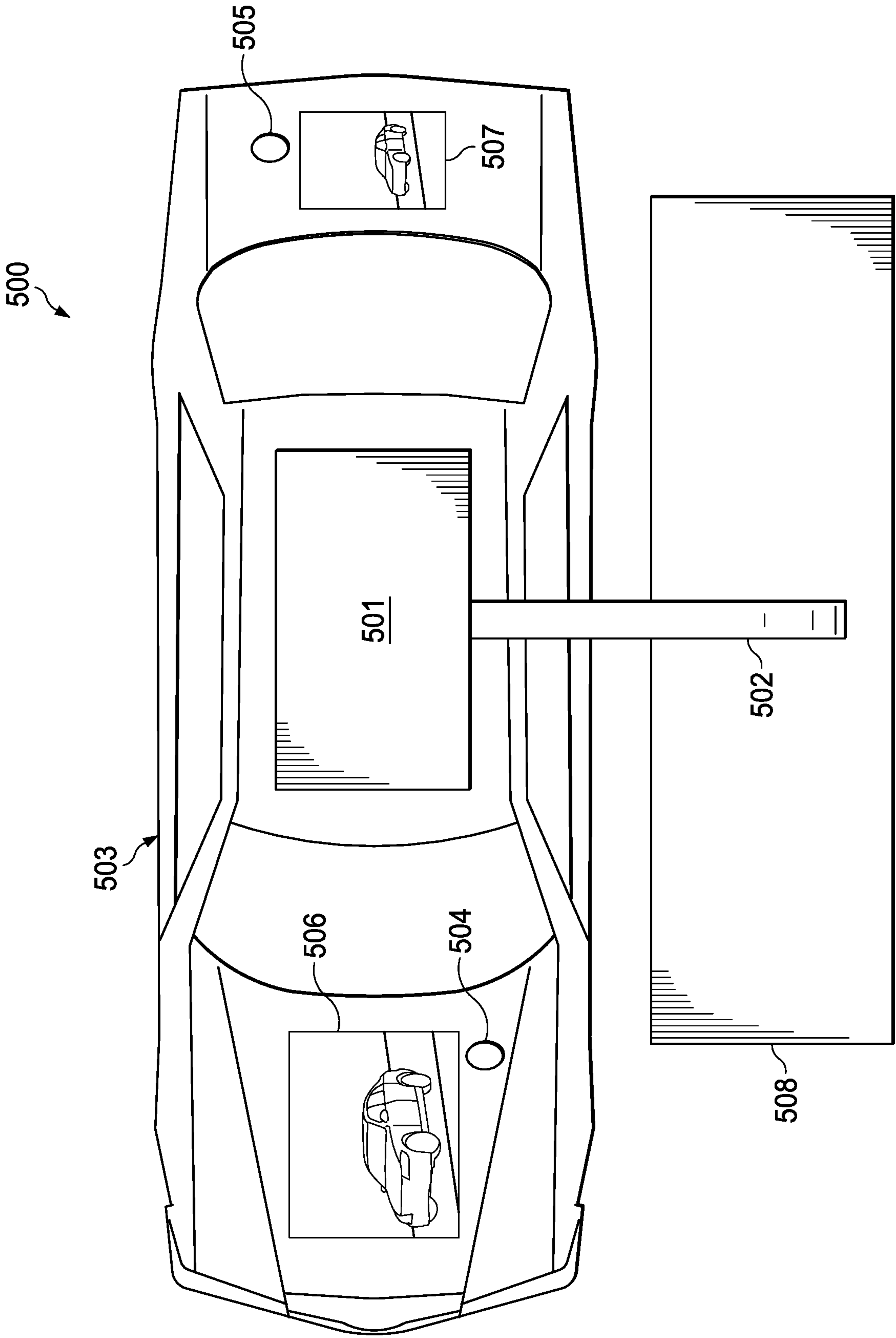


FIG. 5

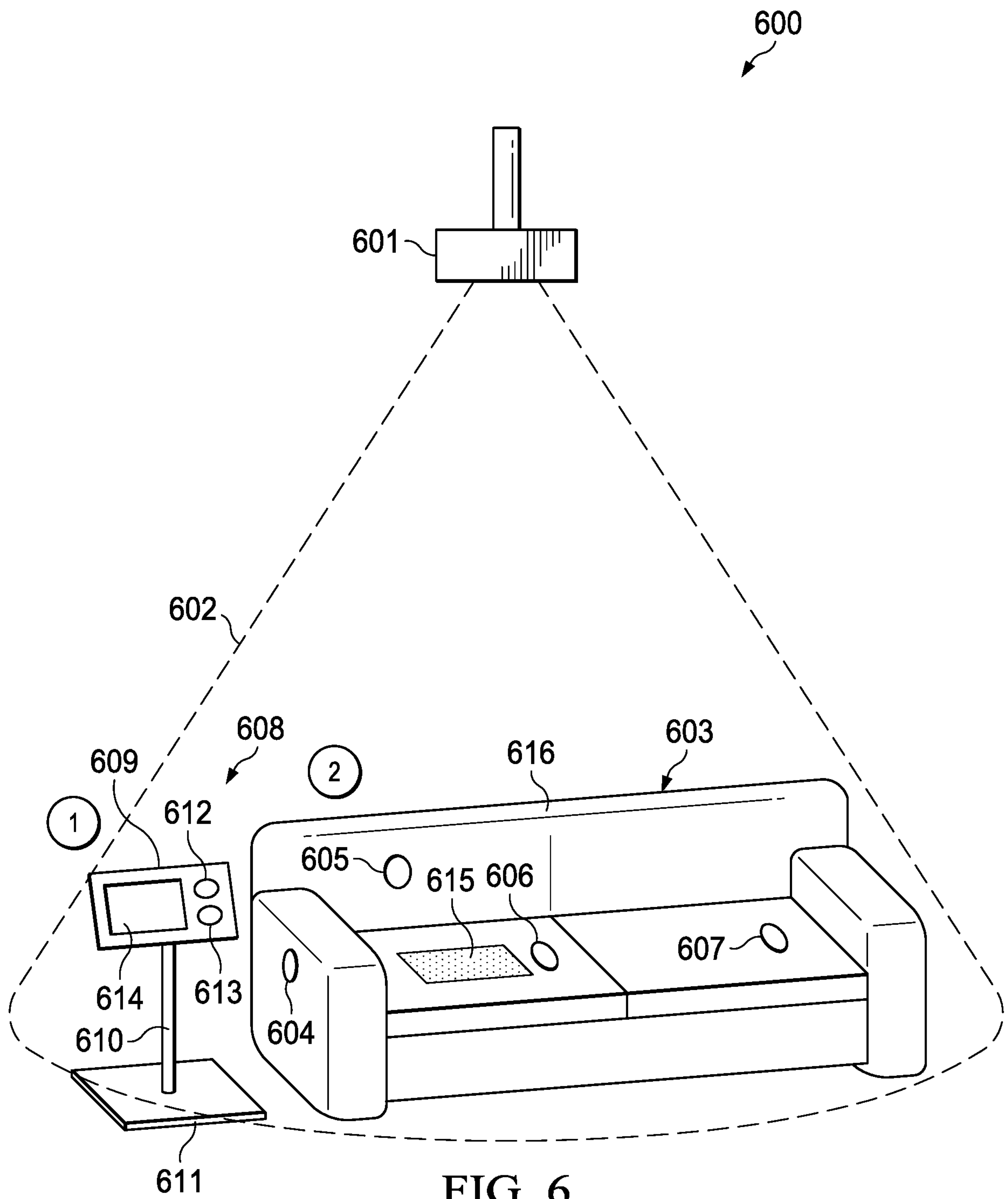
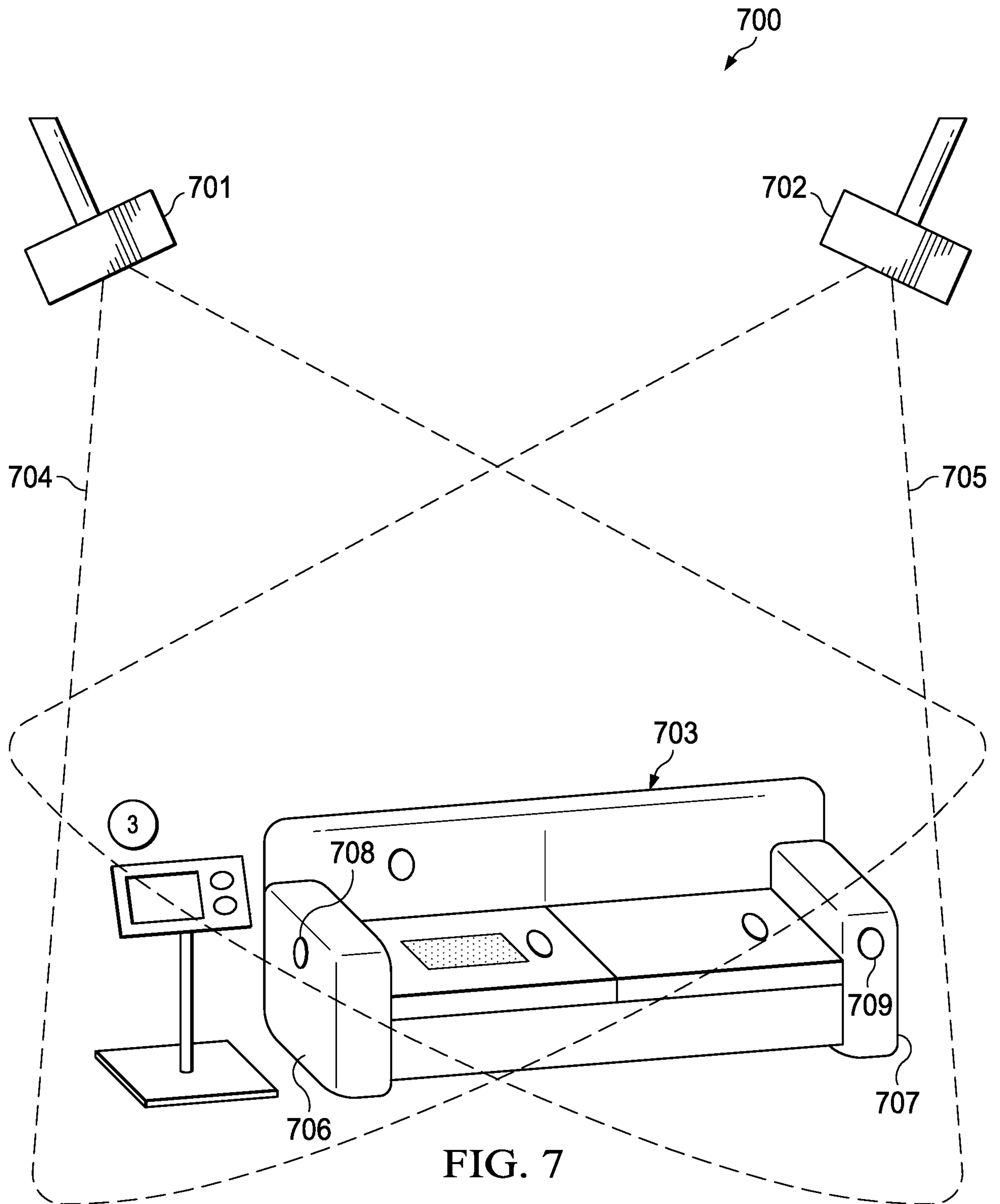


FIG. 6



7/26

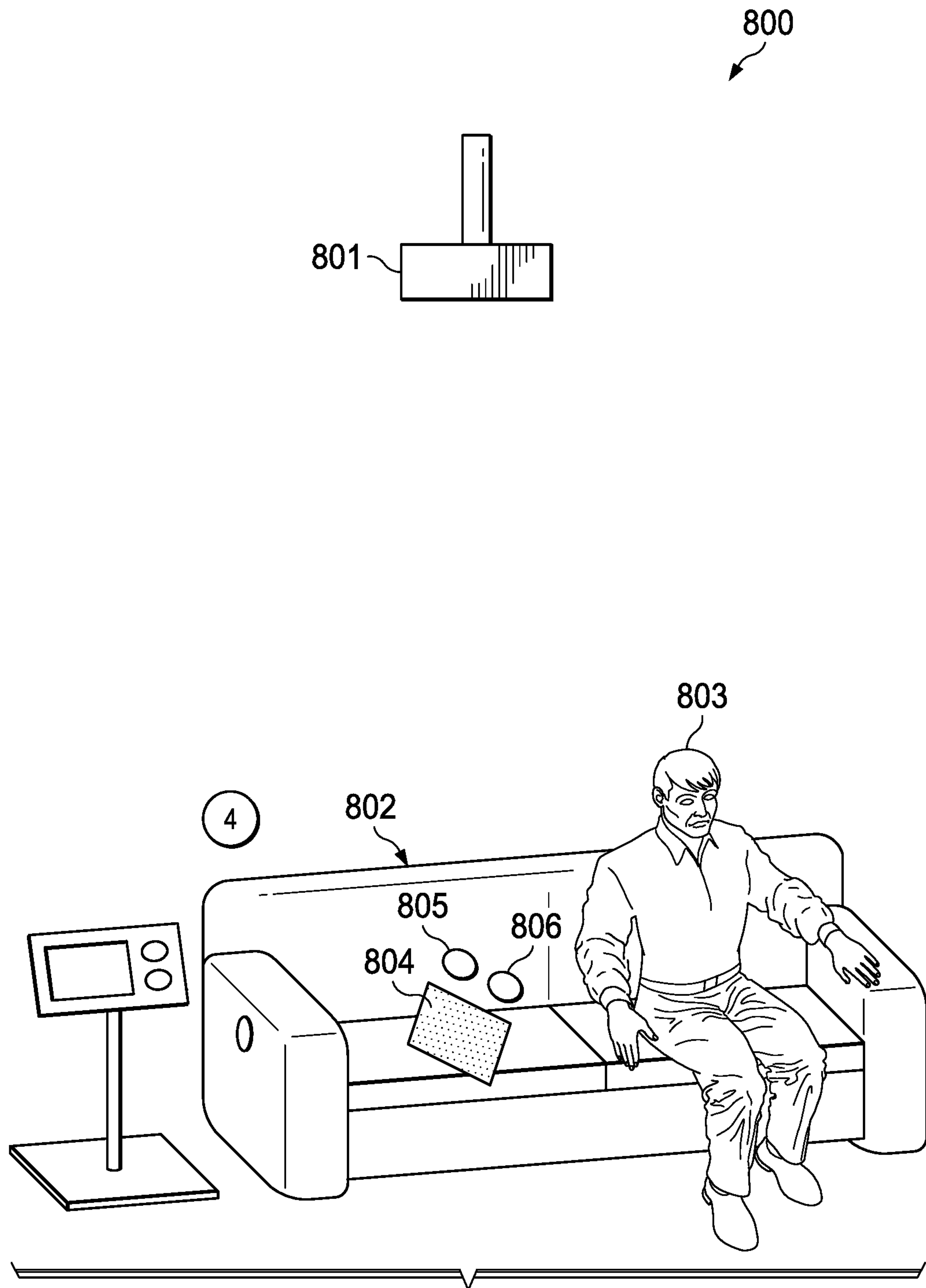


FIG. 8

8/26

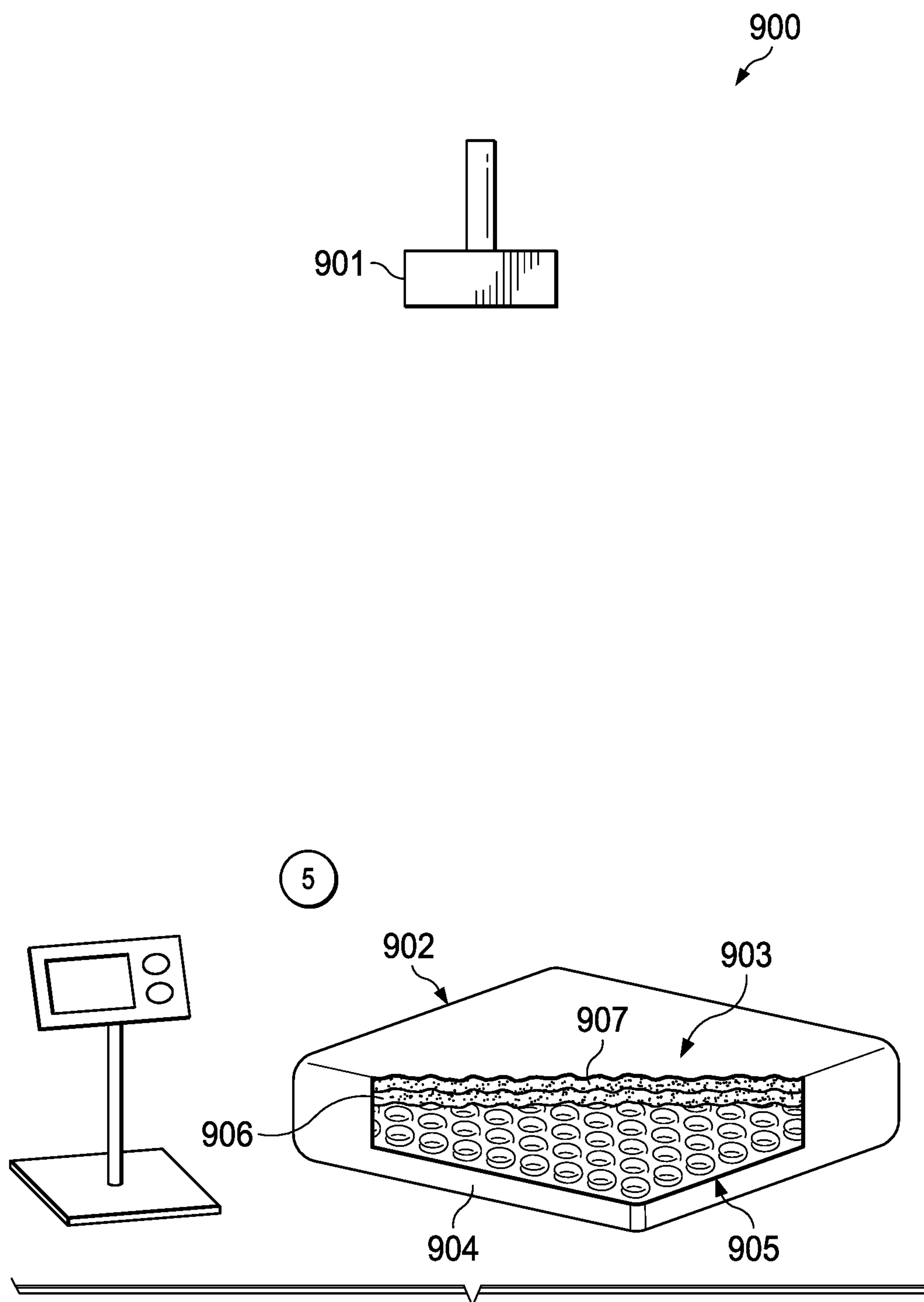


FIG. 9

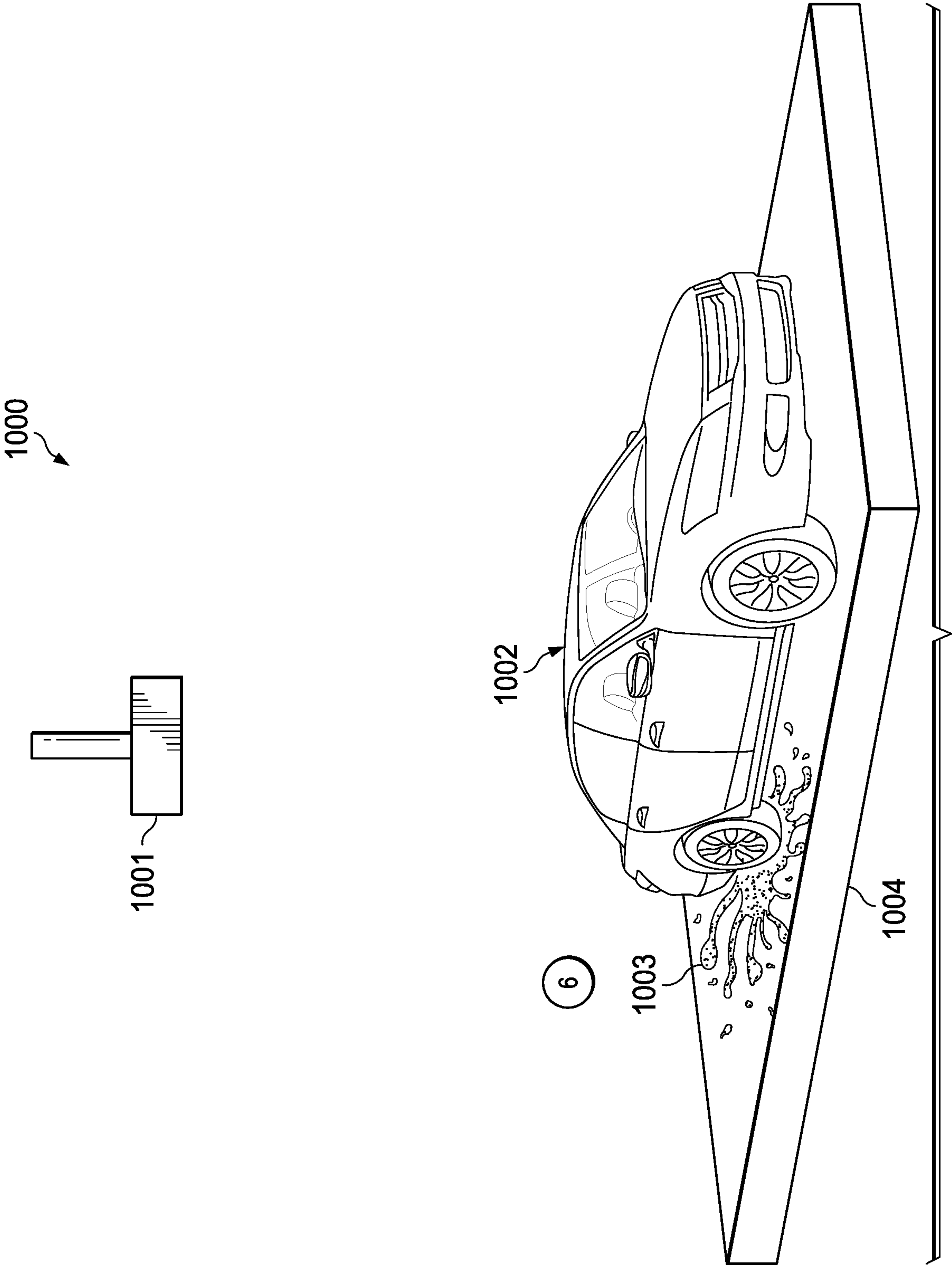
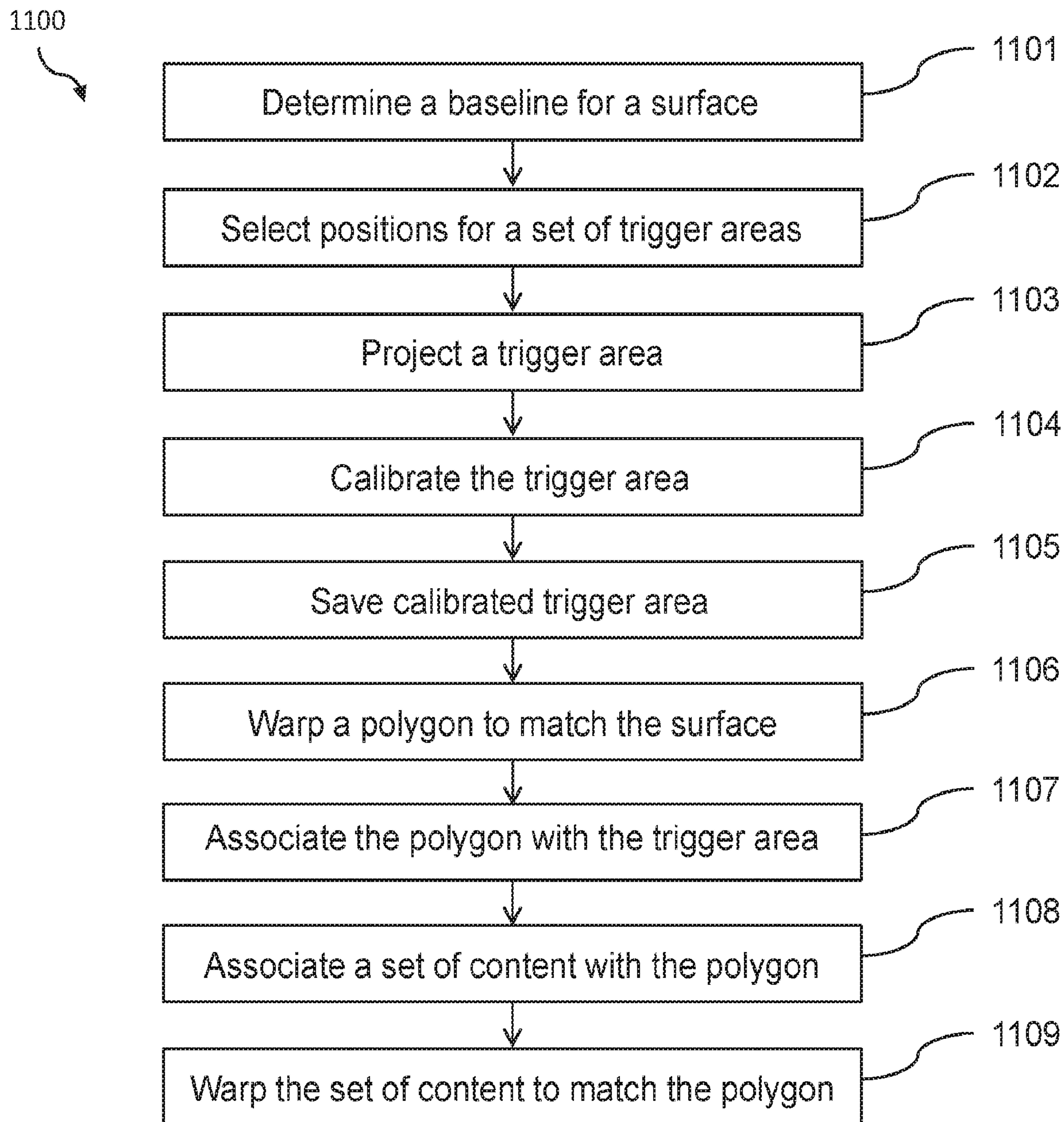


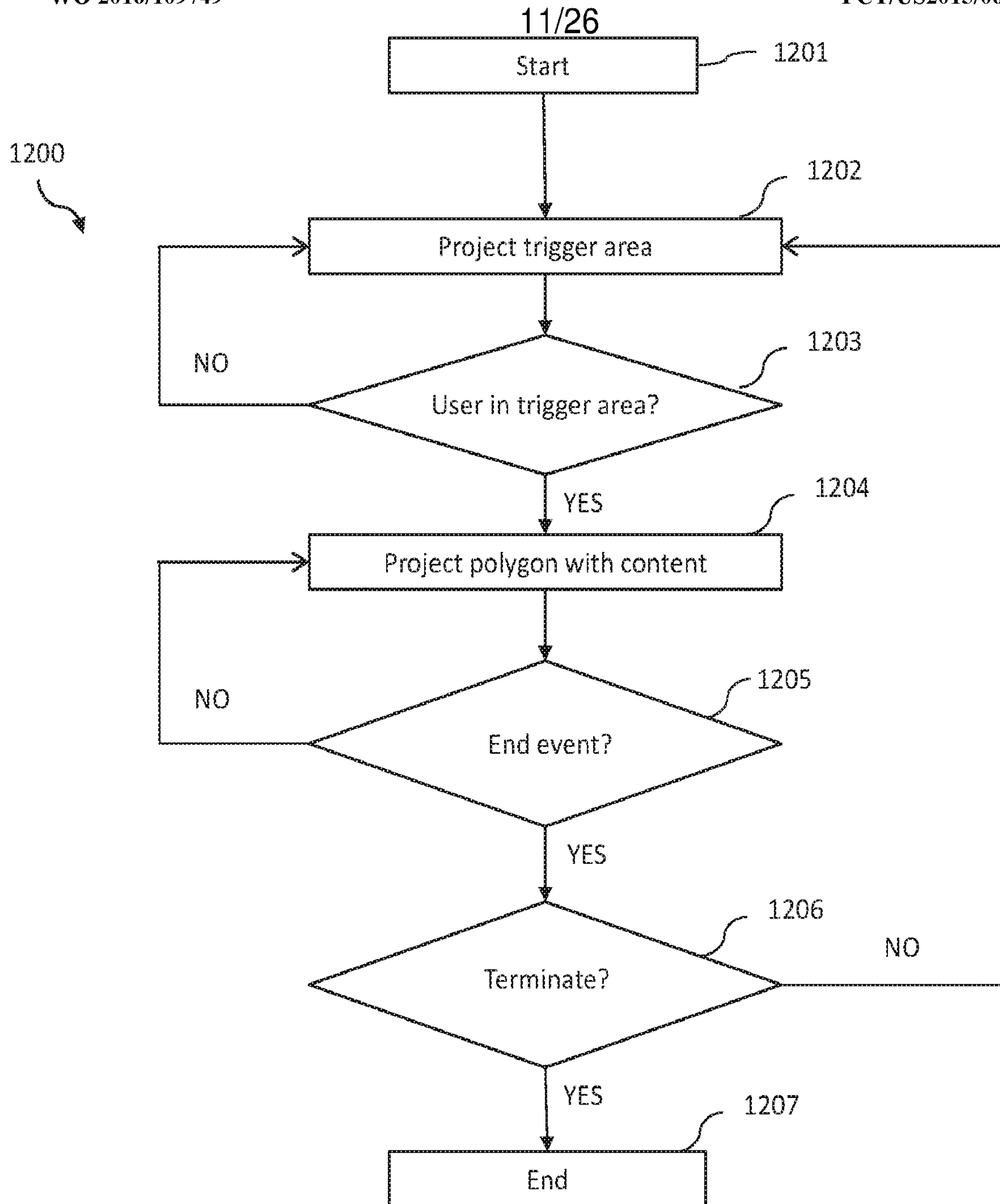
FIG. 10

10/26



Calibration Method

FIG. 11



Run-time

FIG. 12

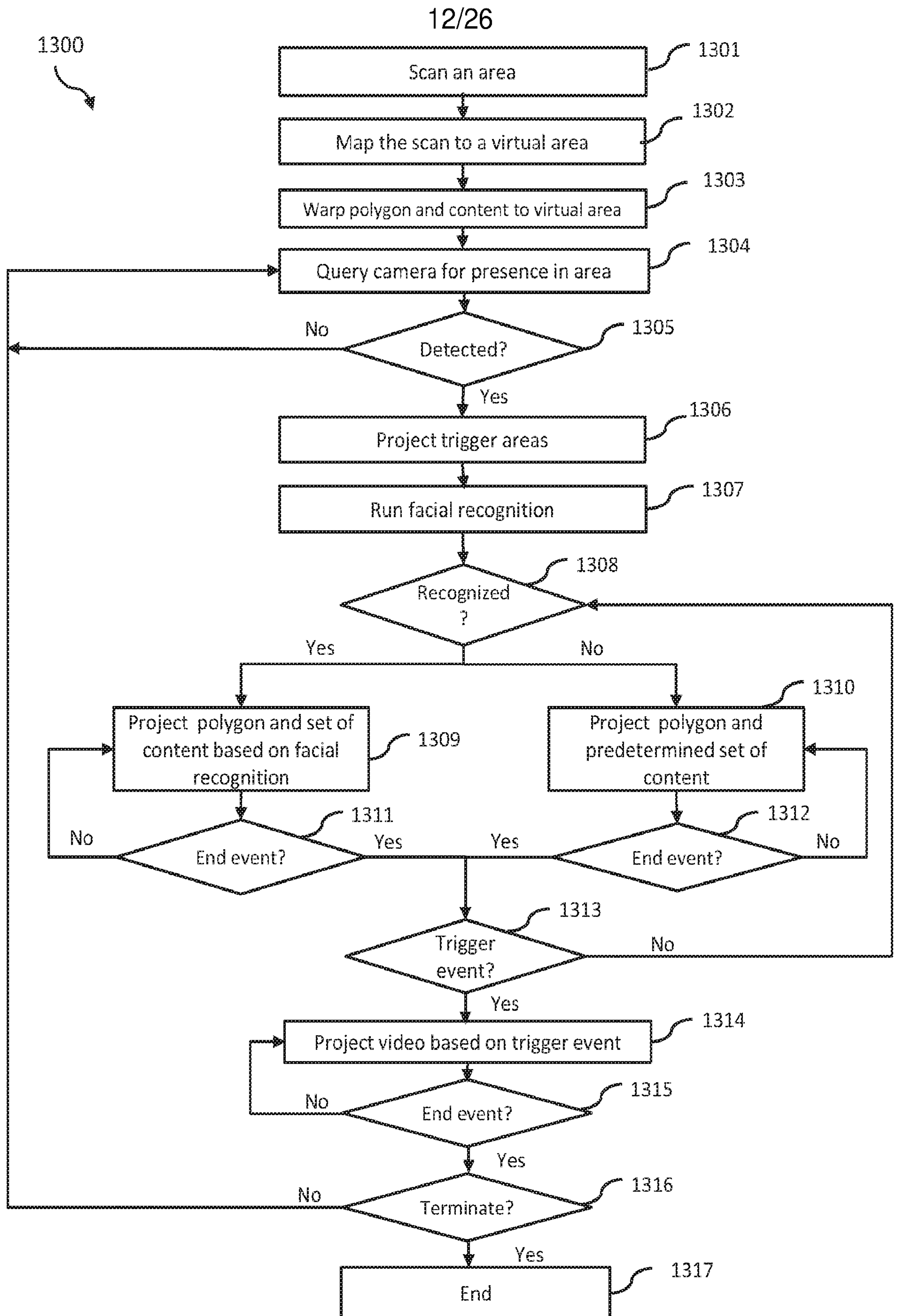
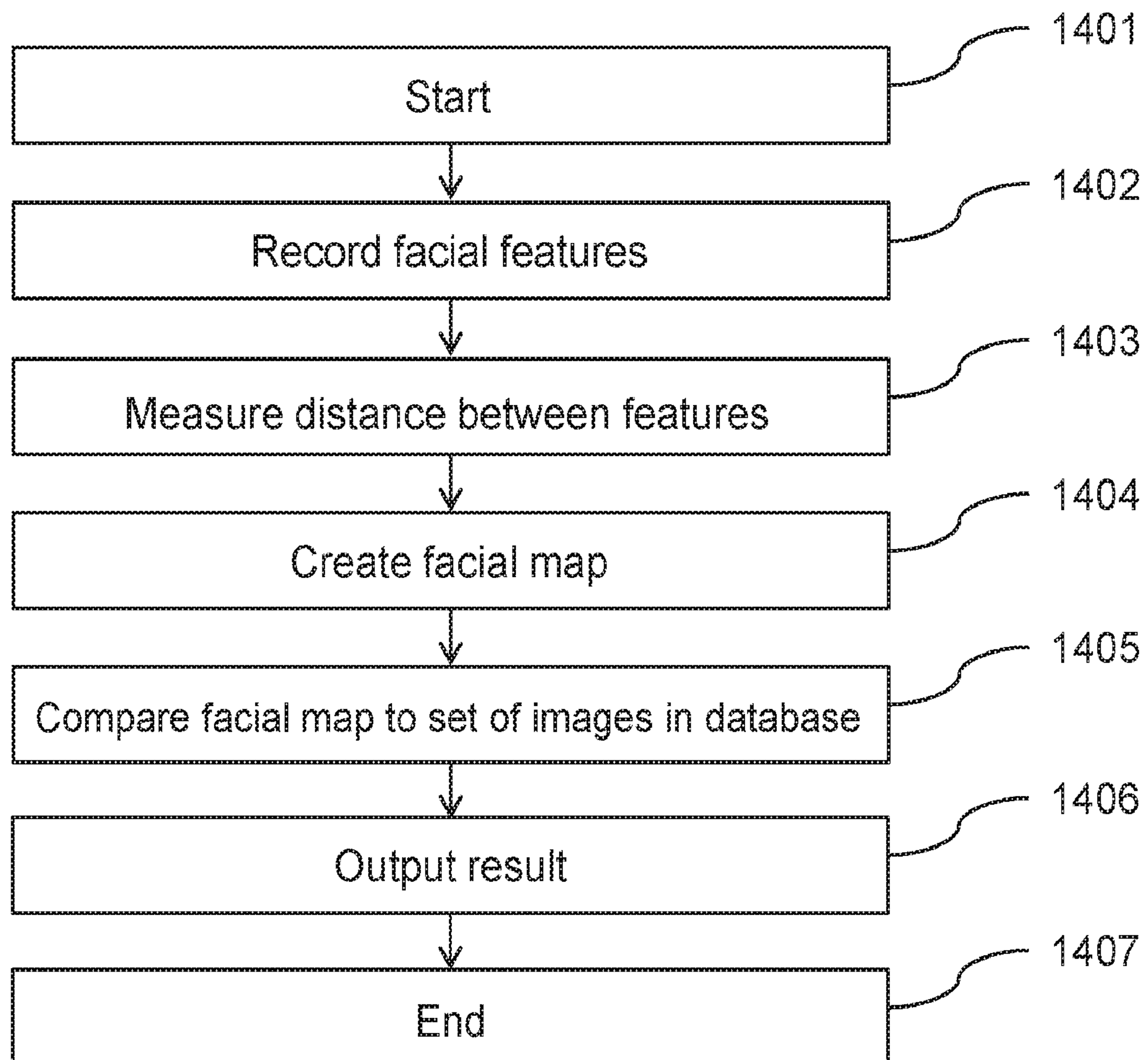


FIG. 13

13/26

1400



Facial Recognition
Routine

FIG. 14

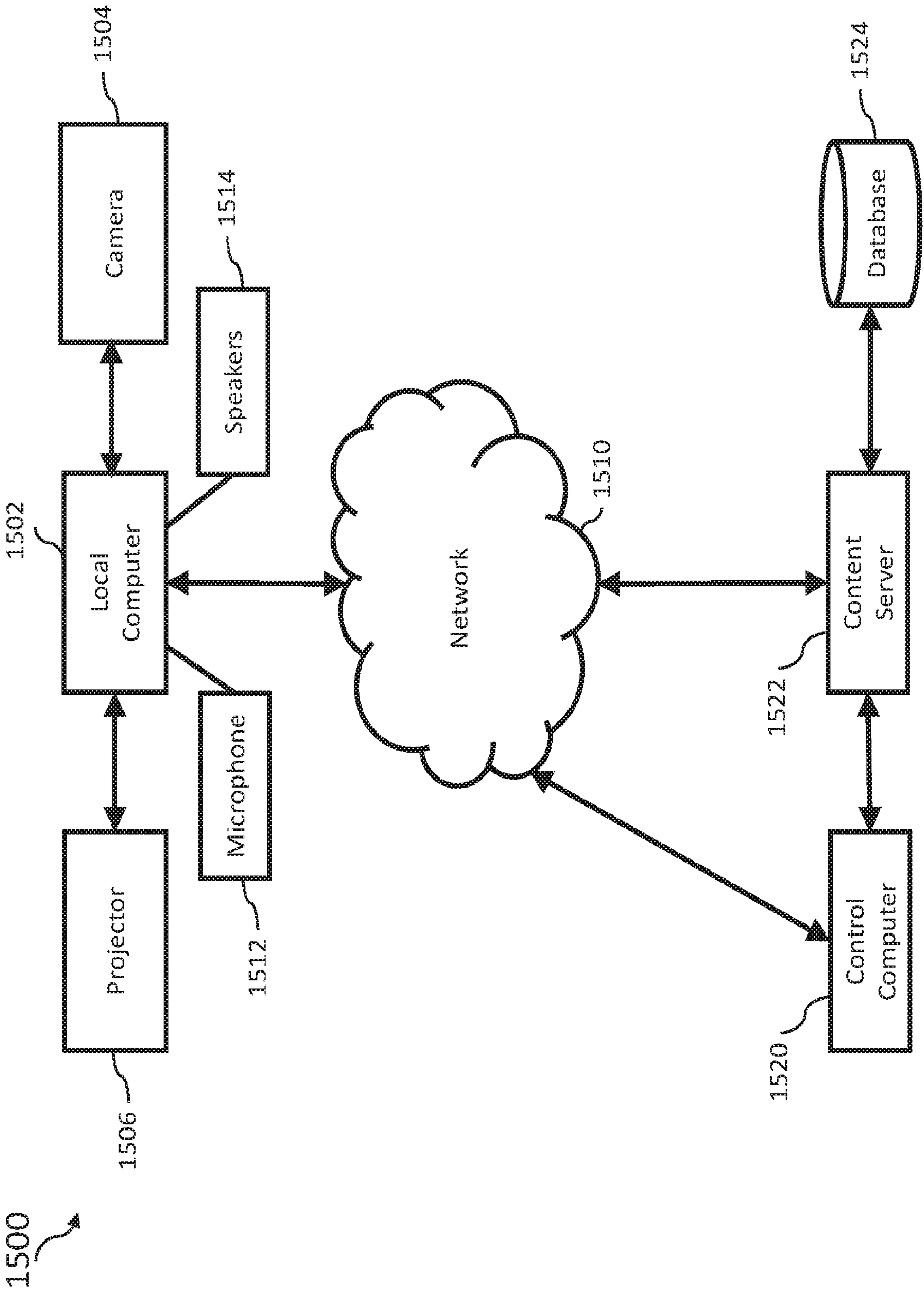


FIG. 15

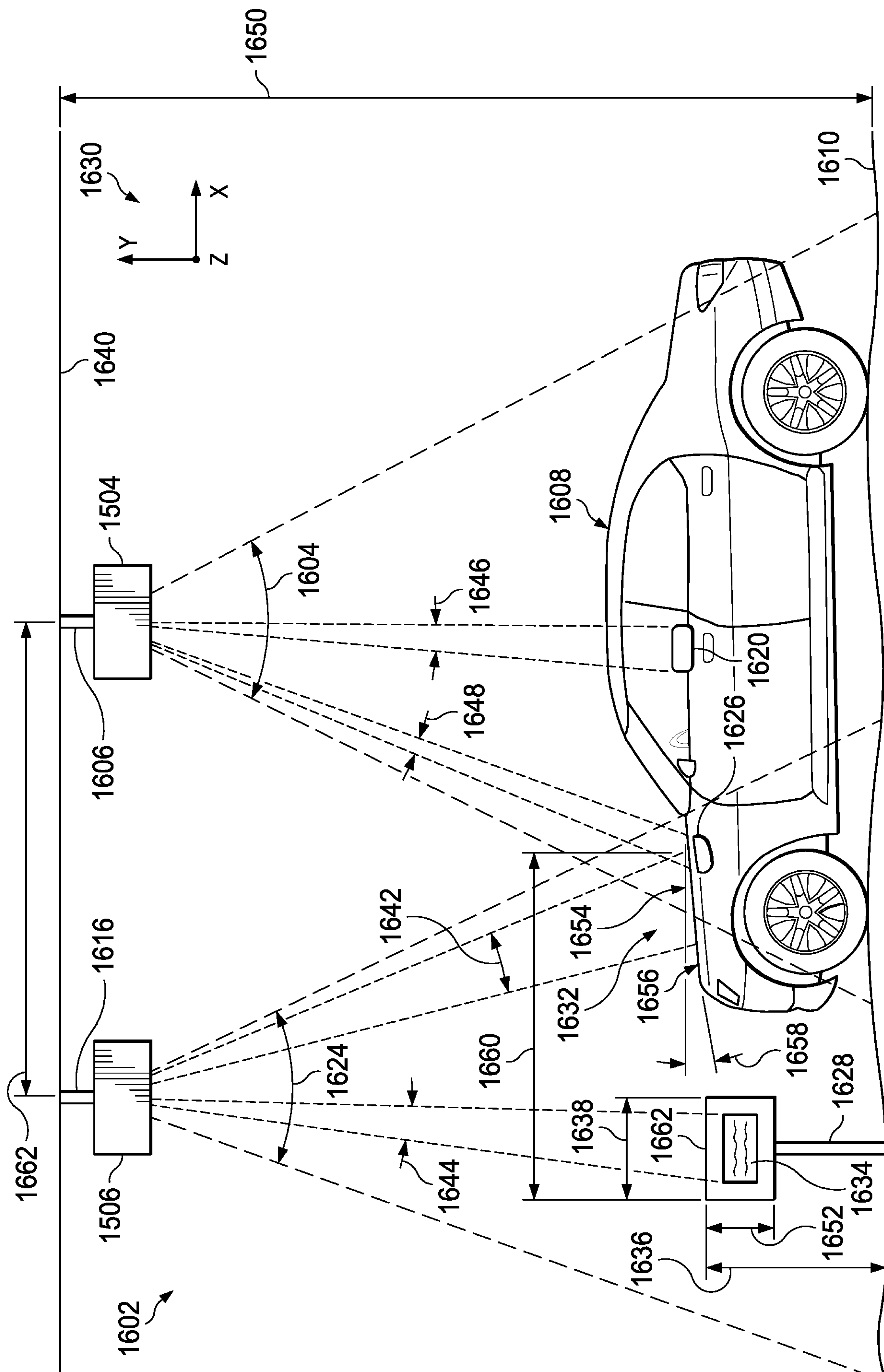


FIG. 16A

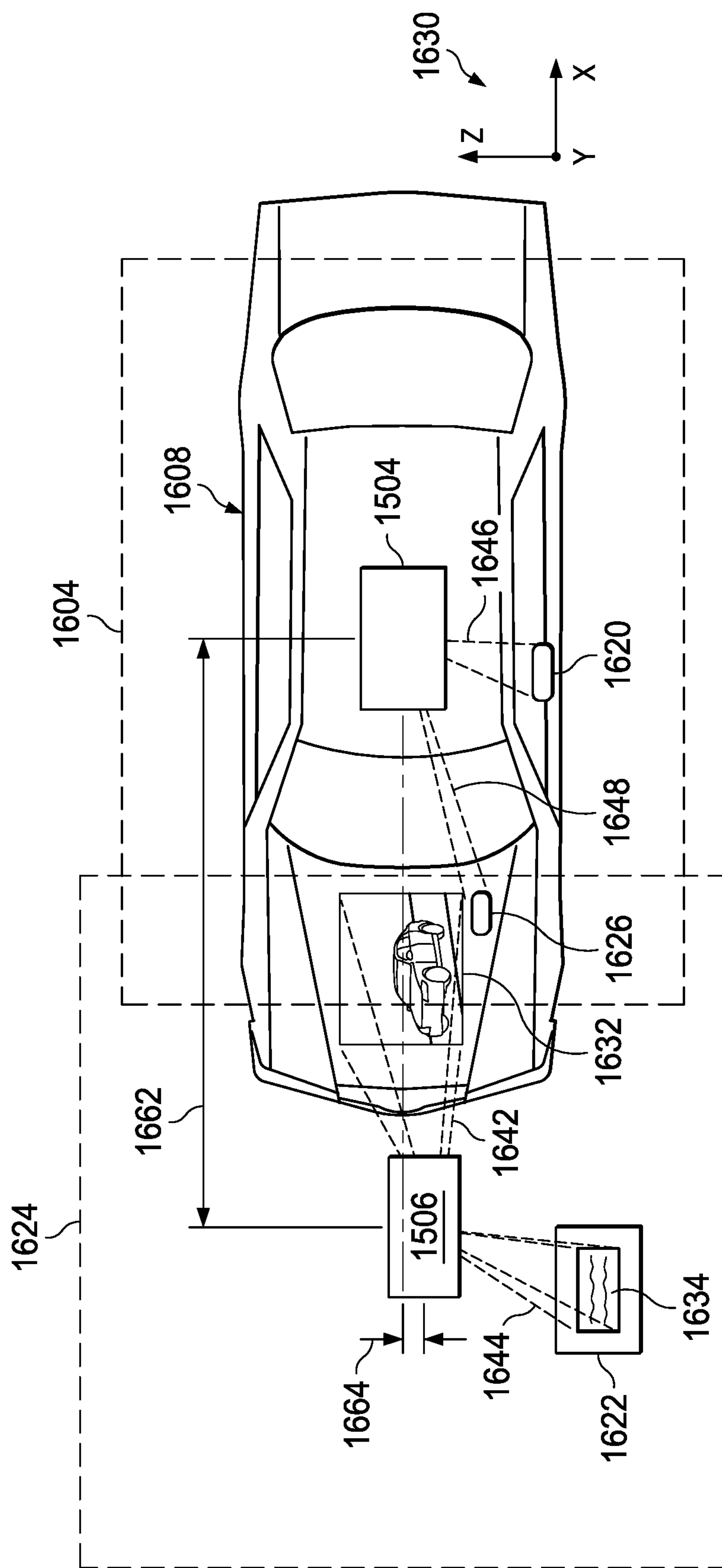


FIG. 16B

1654

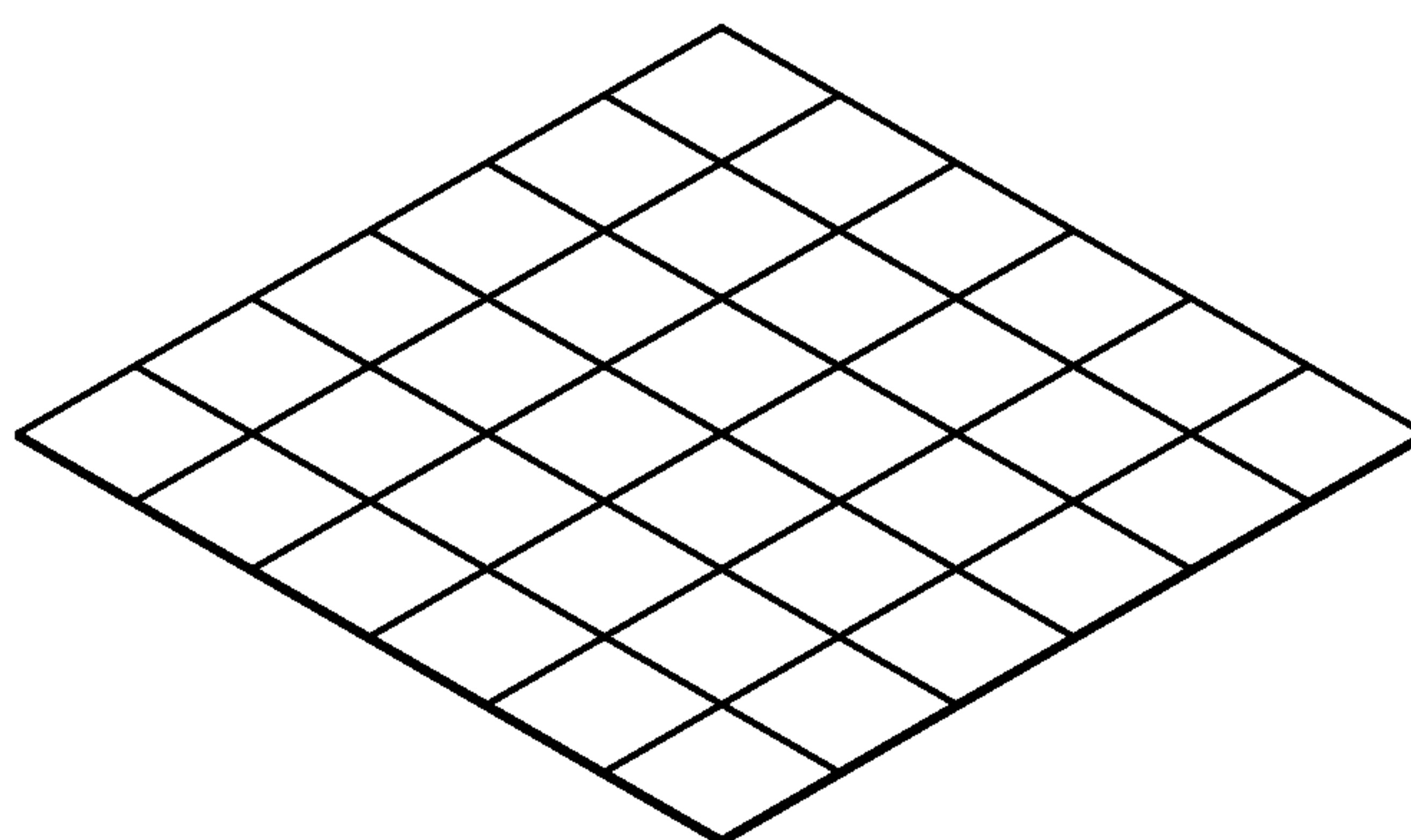


FIG. 16C

1656

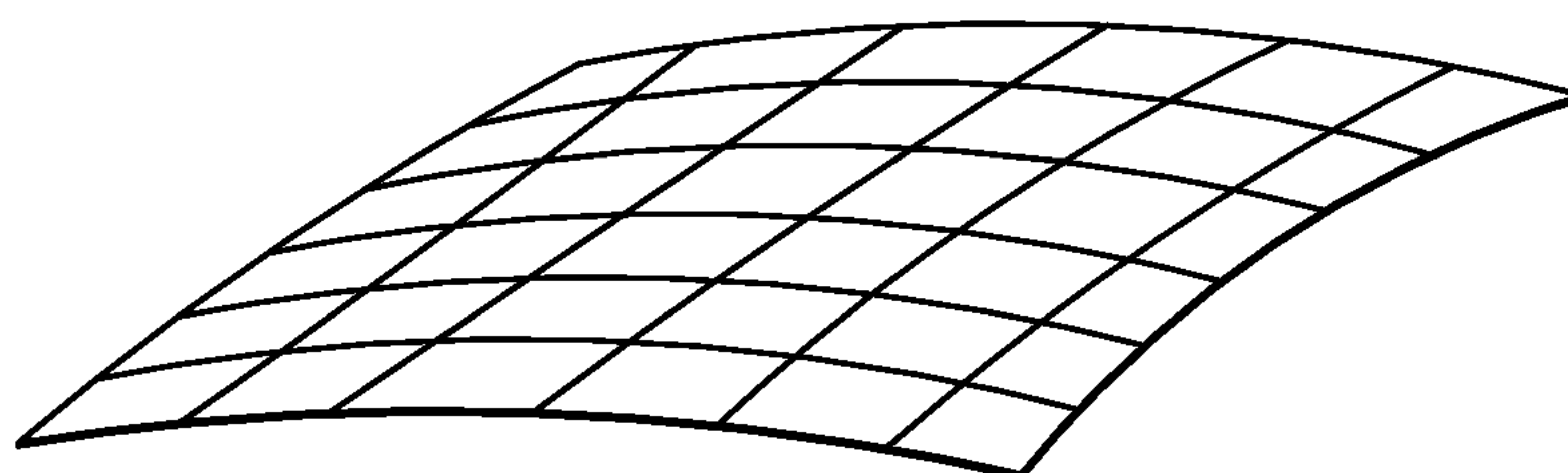


FIG. 16D

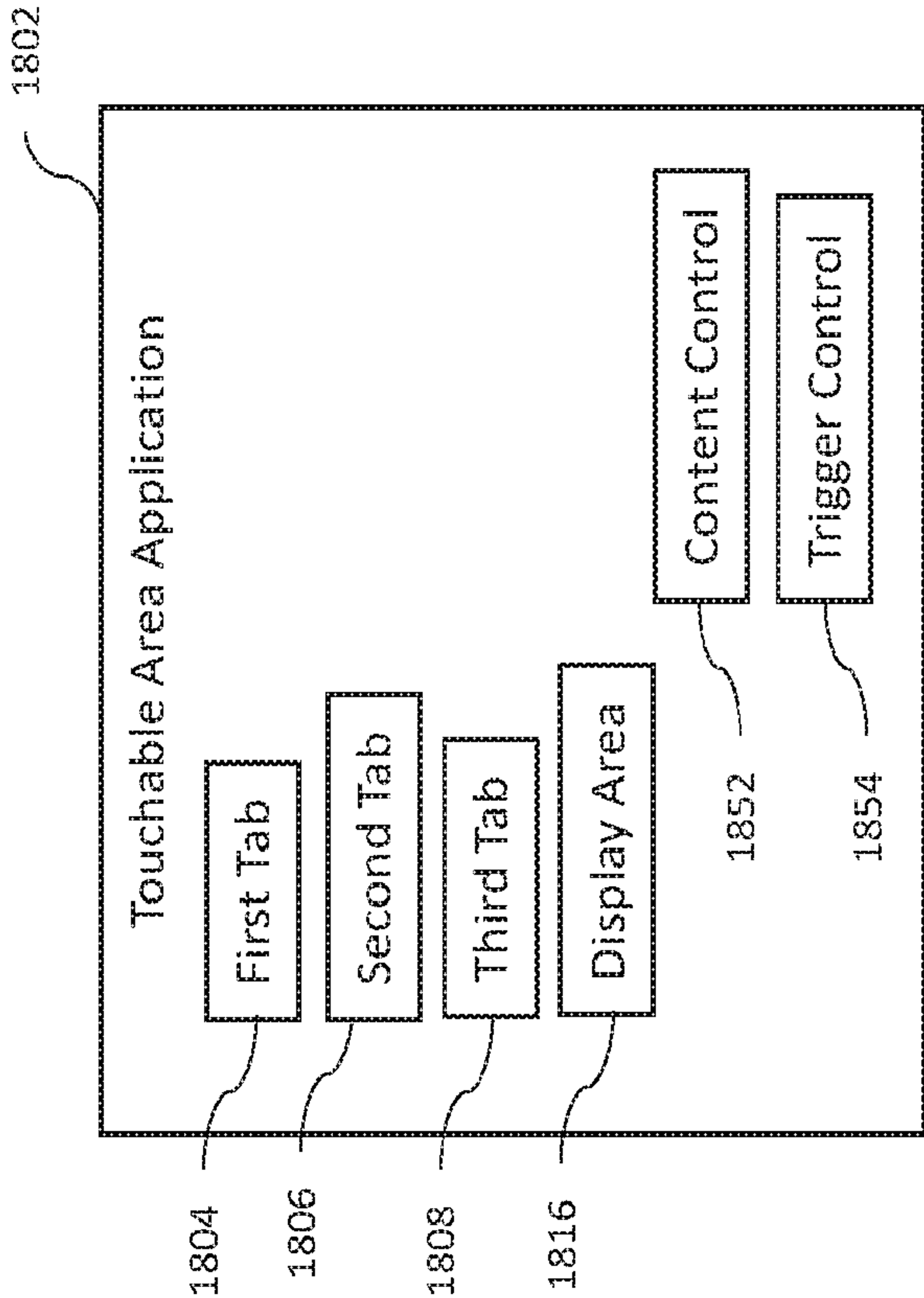


FIG. 18A

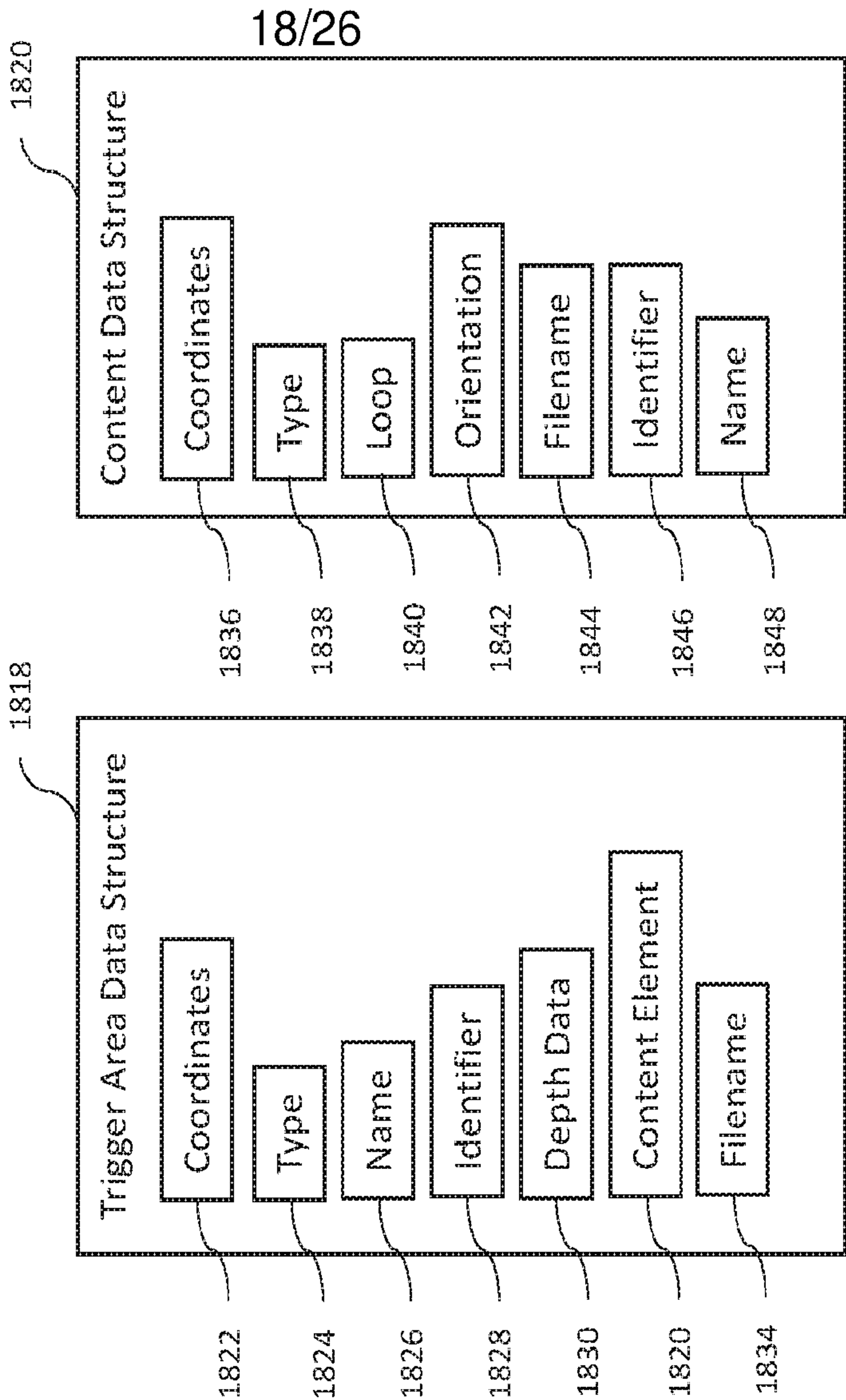


FIG. 18B

FIG. 18C

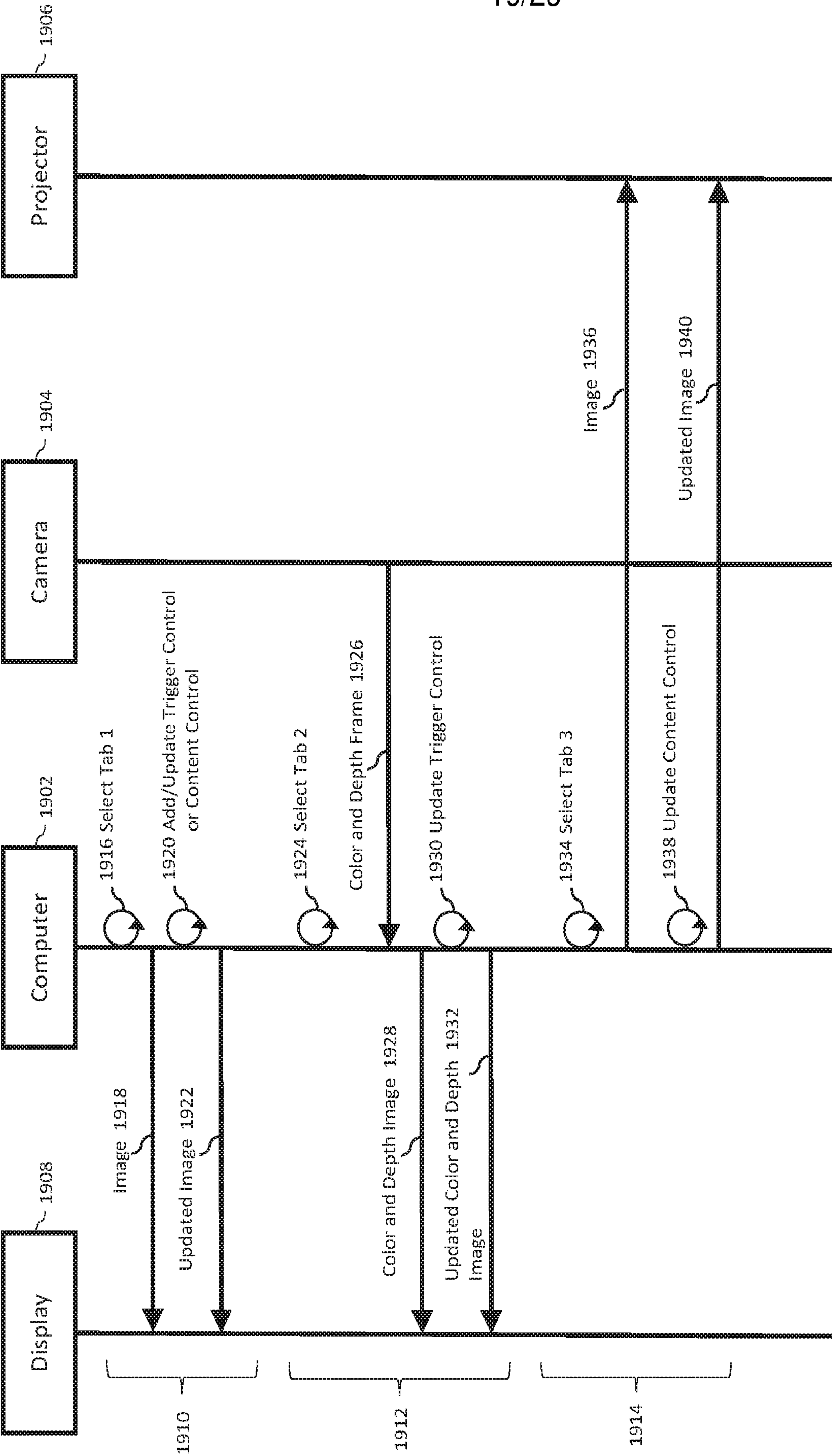


FIG. 19A

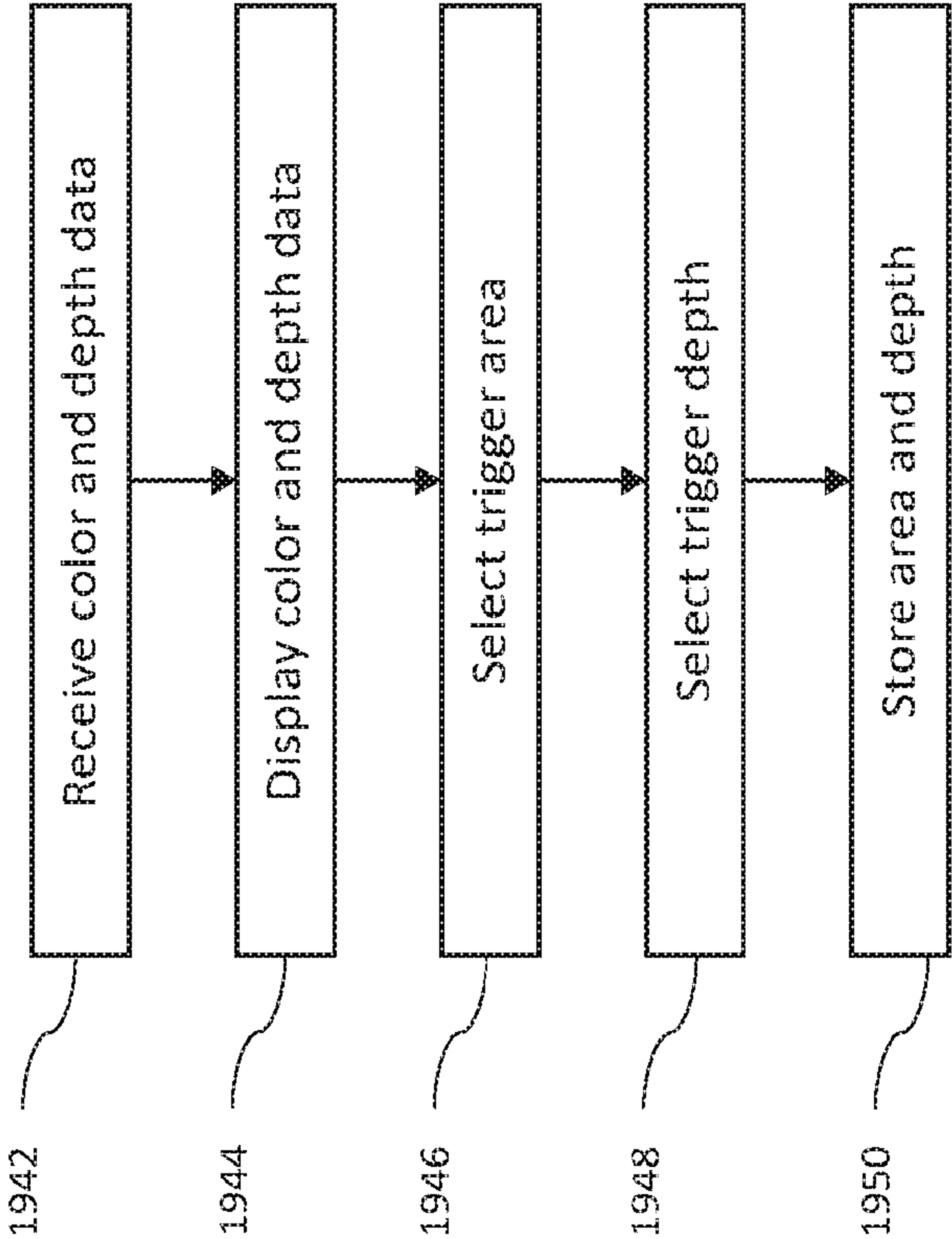


FIG. 19B

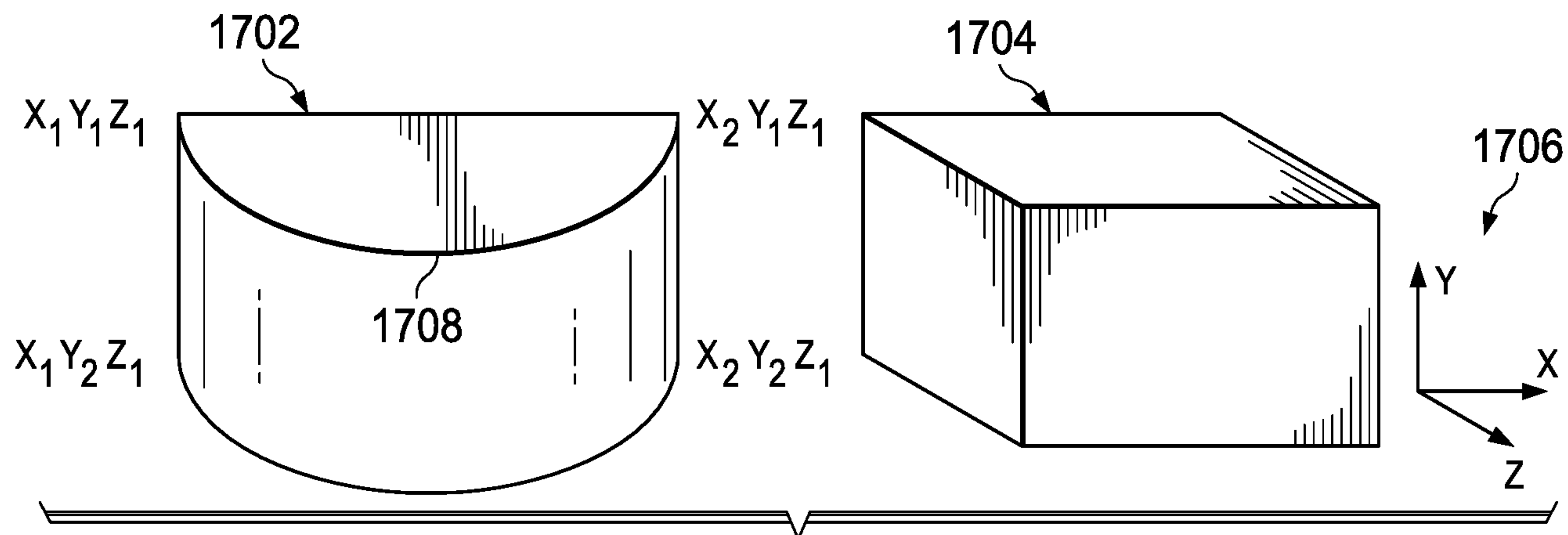


FIG. 17

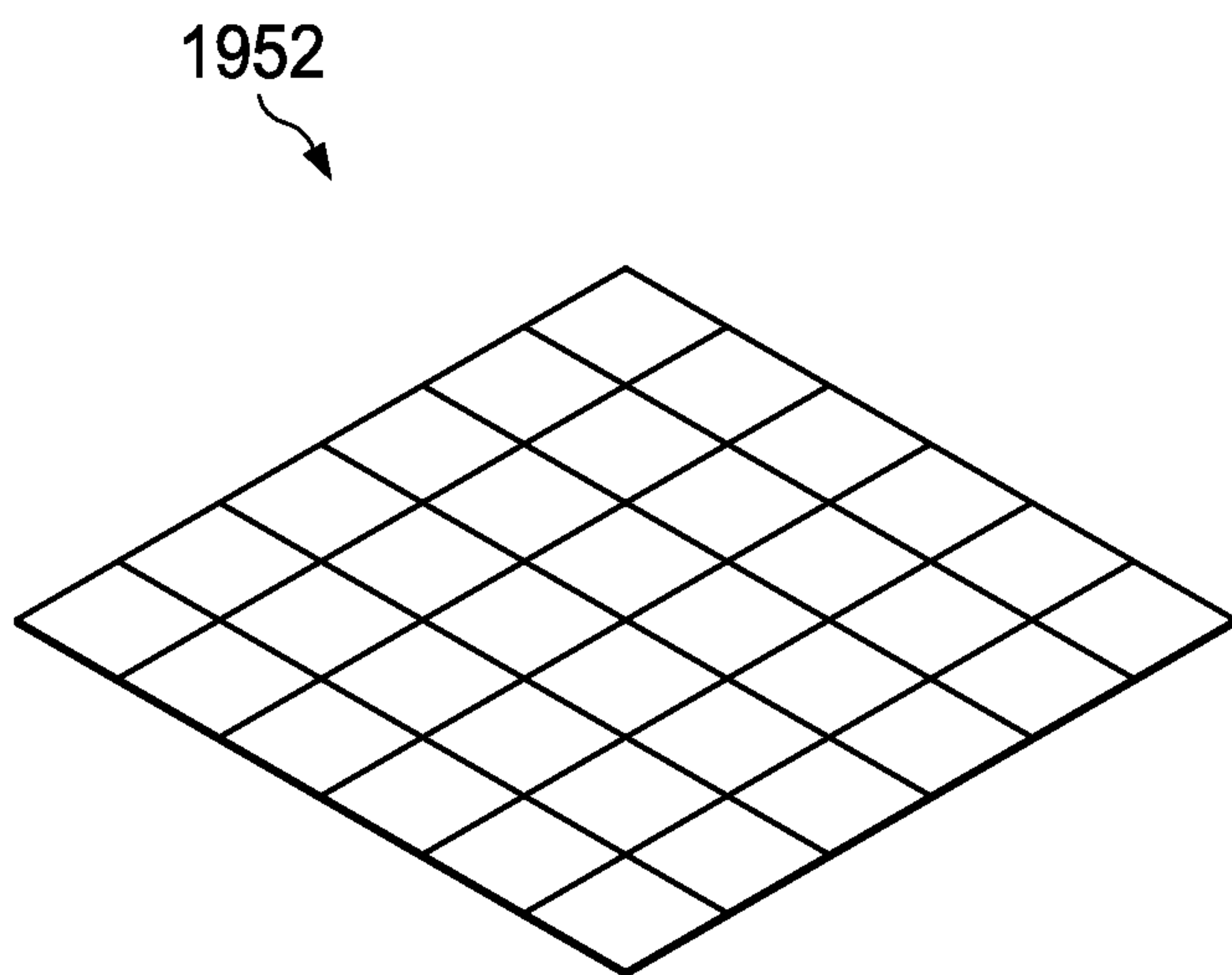


FIG. 19C

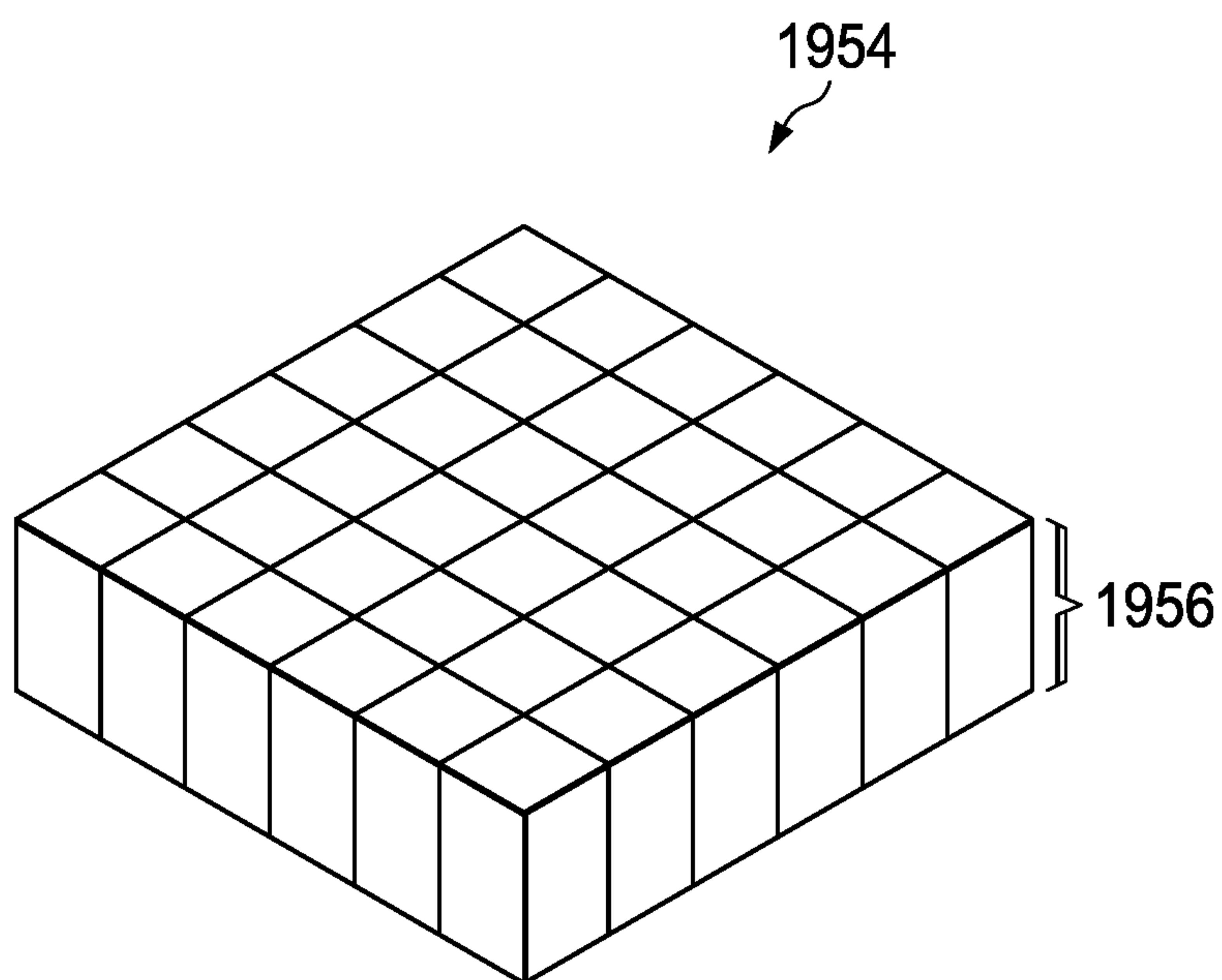


FIG. 19D

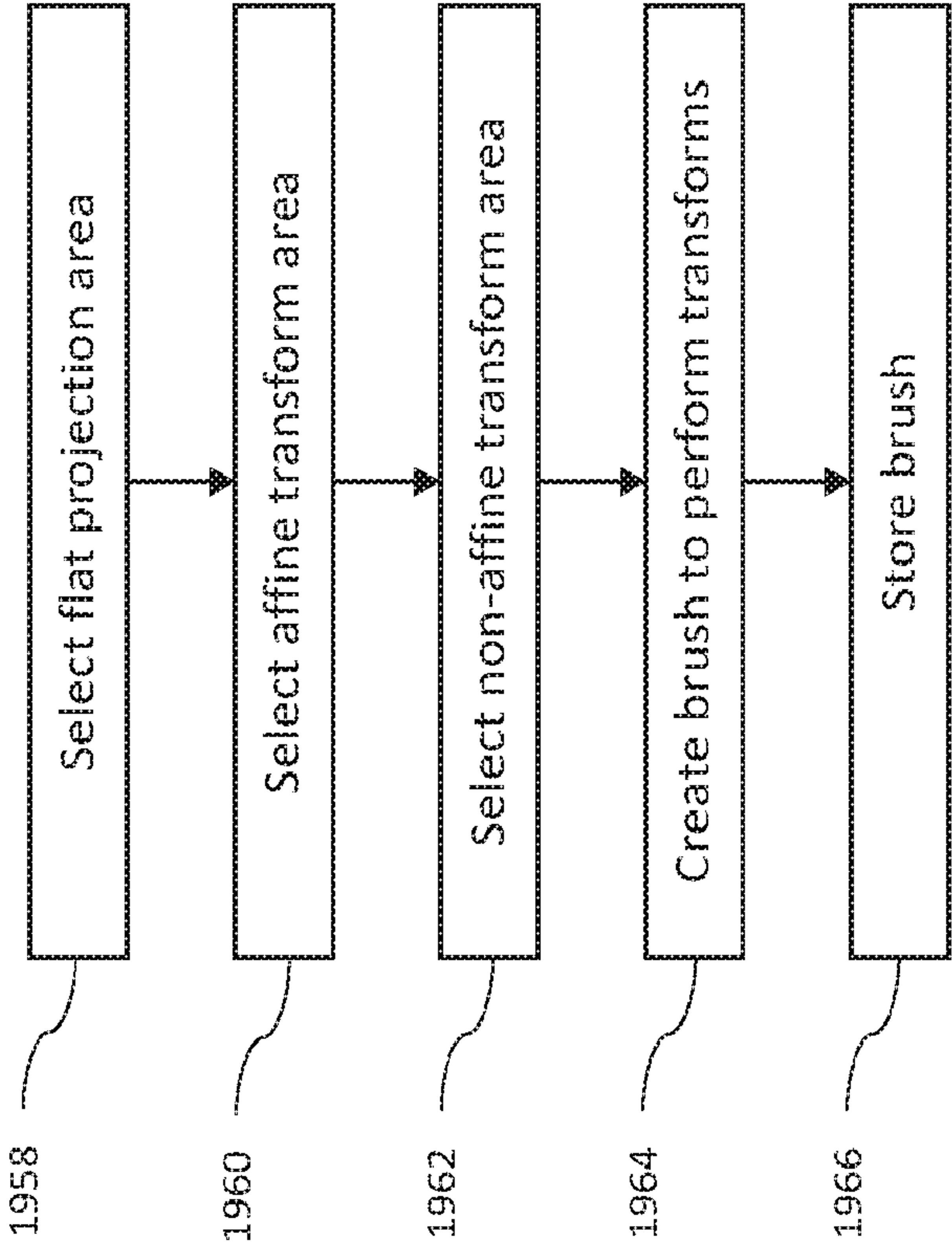
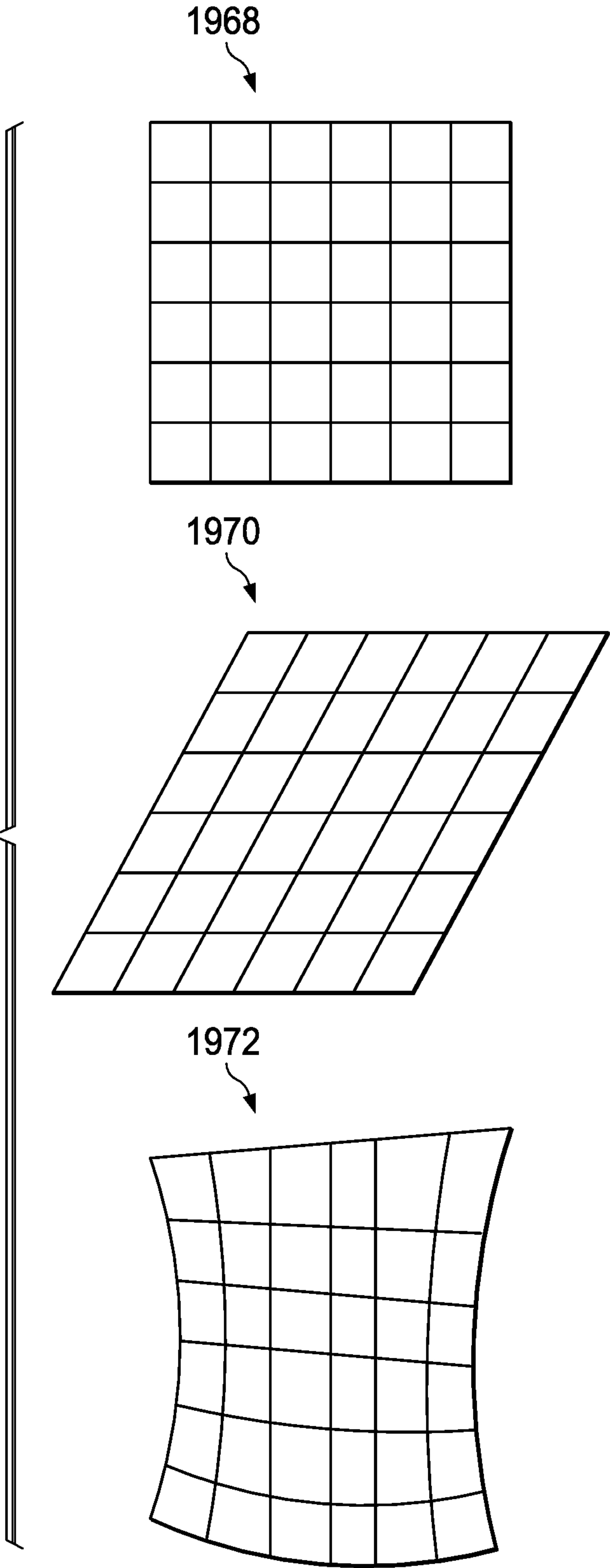


FIG. 19E

FIG. 19F



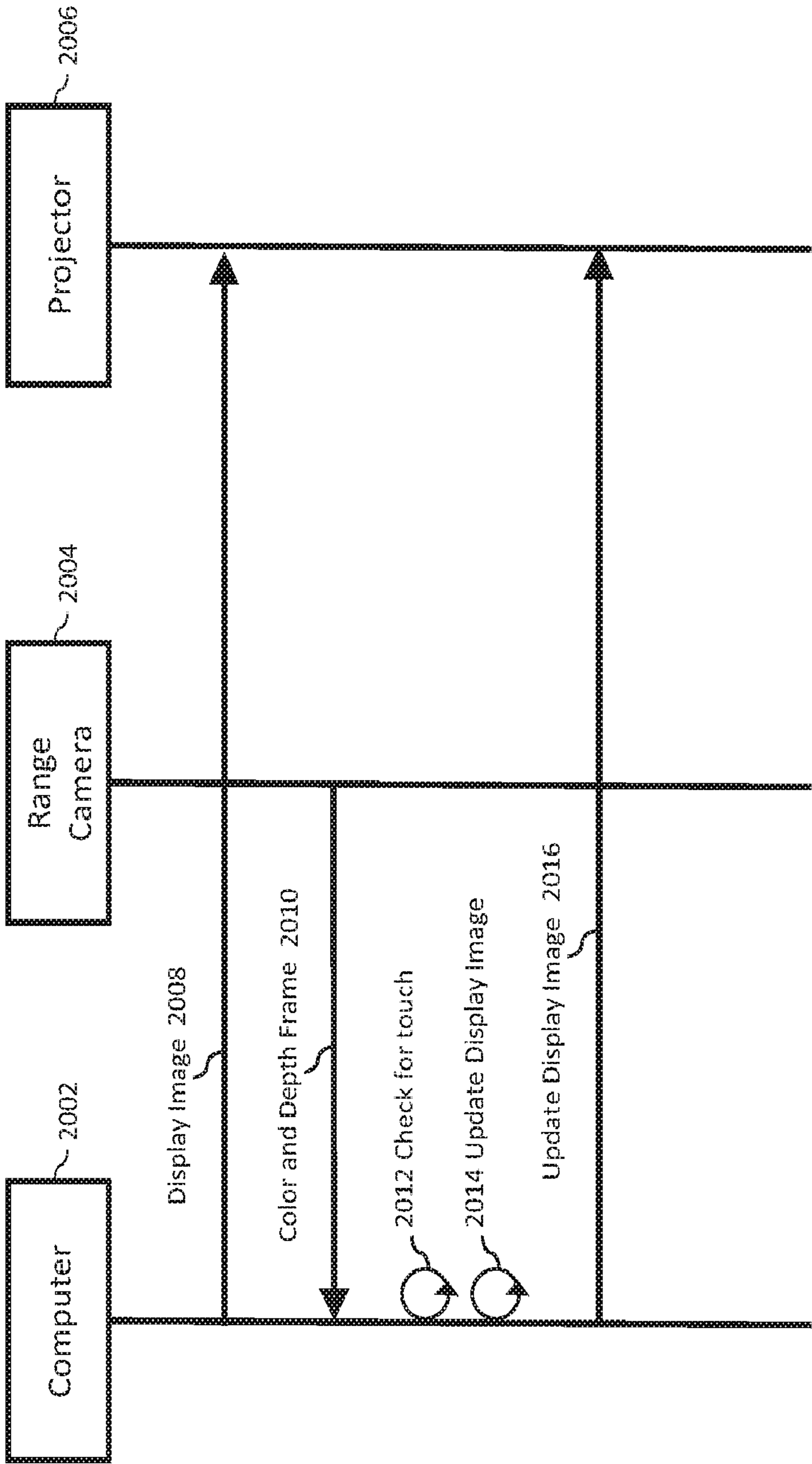


FIG. 20A

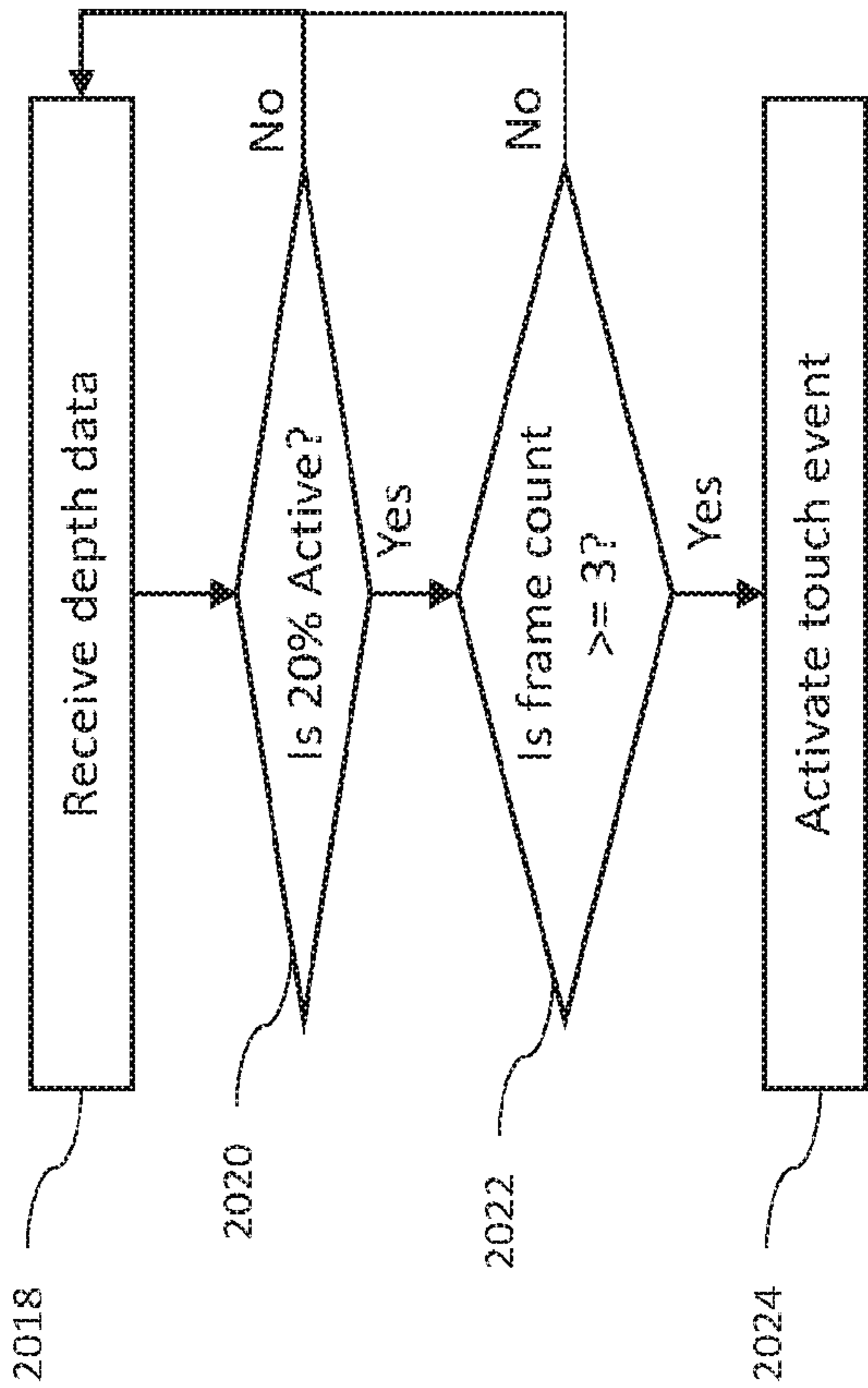


FIG. 20B

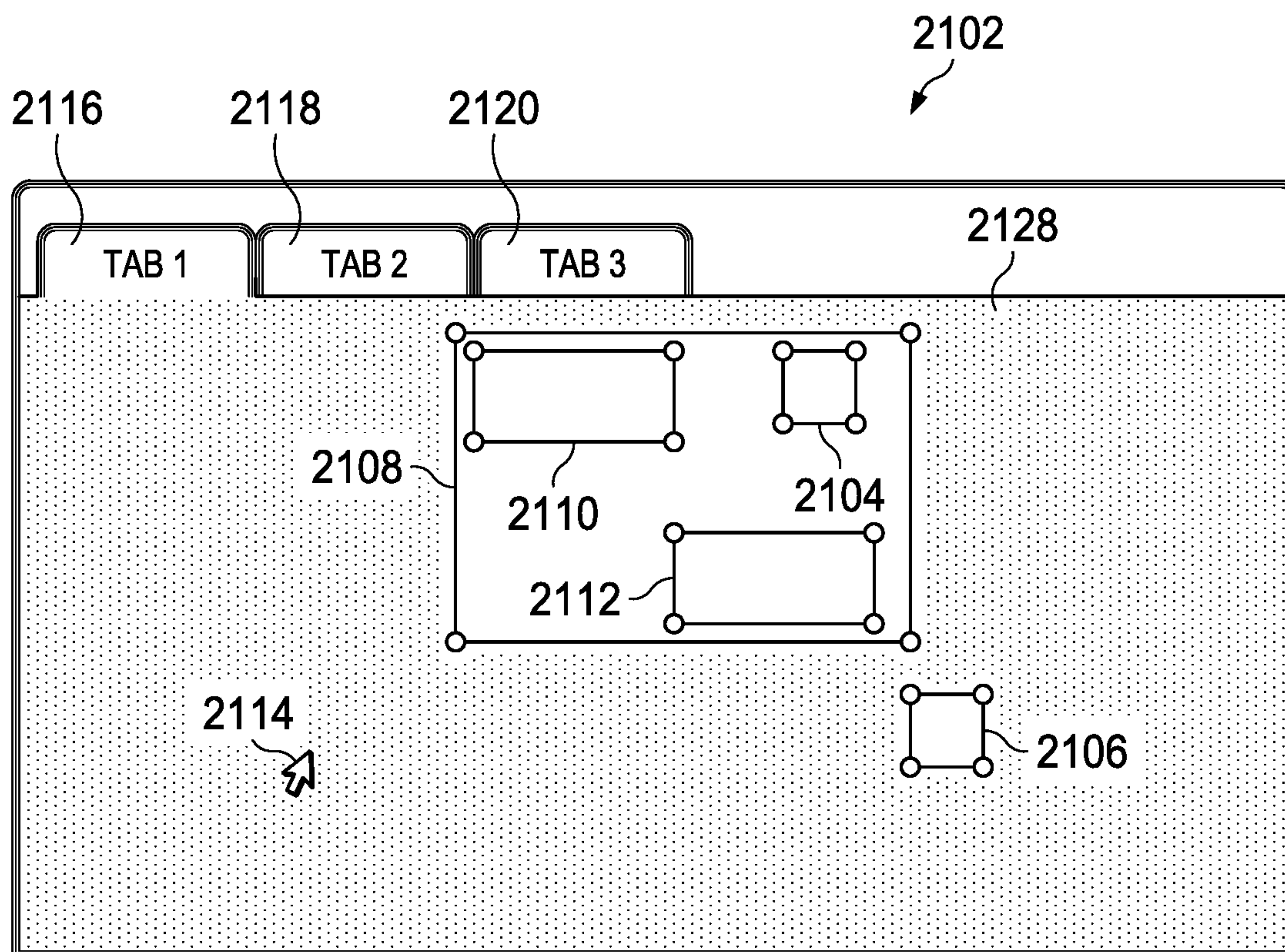


FIG. 21A

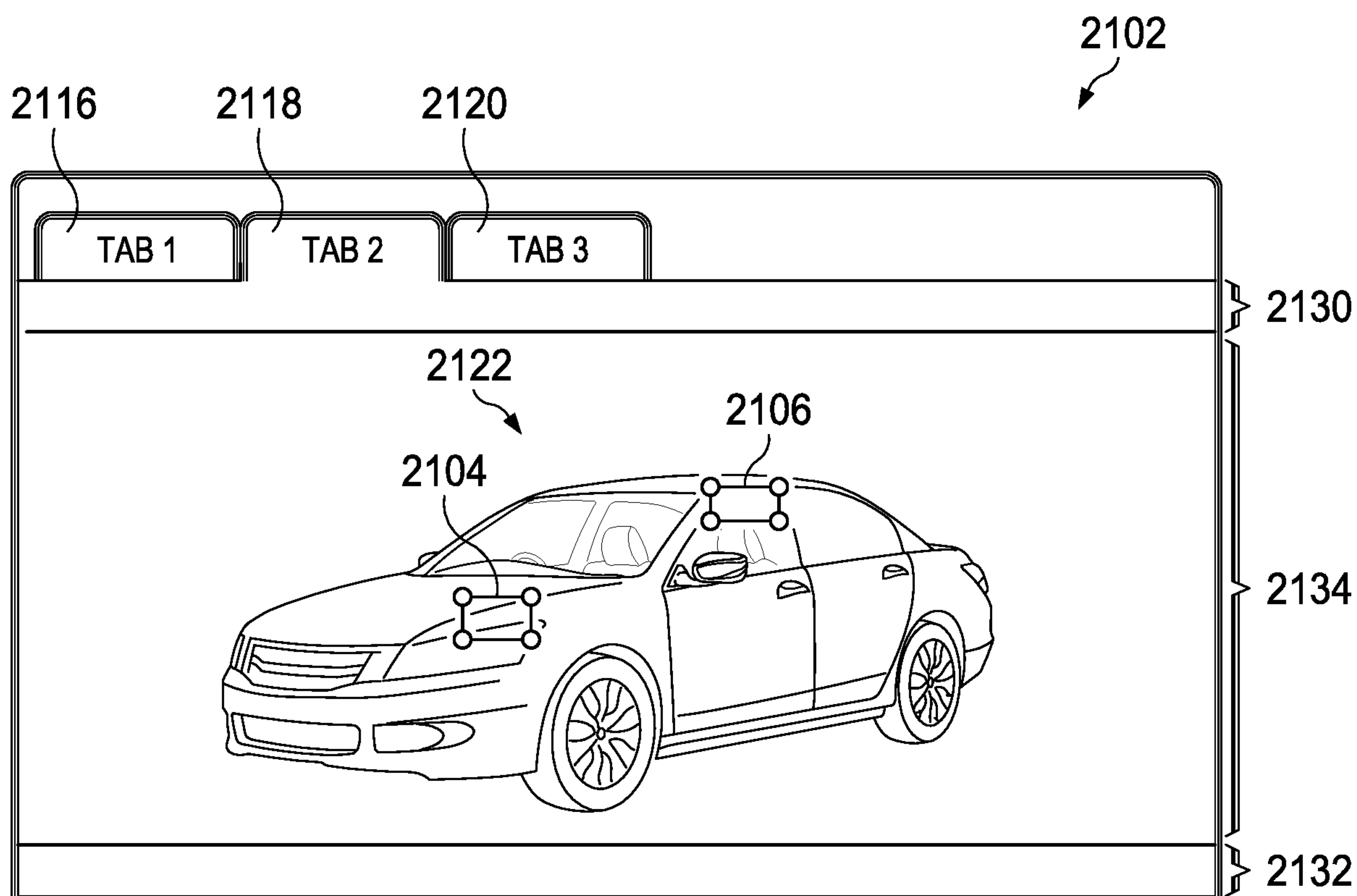


FIG. 21B

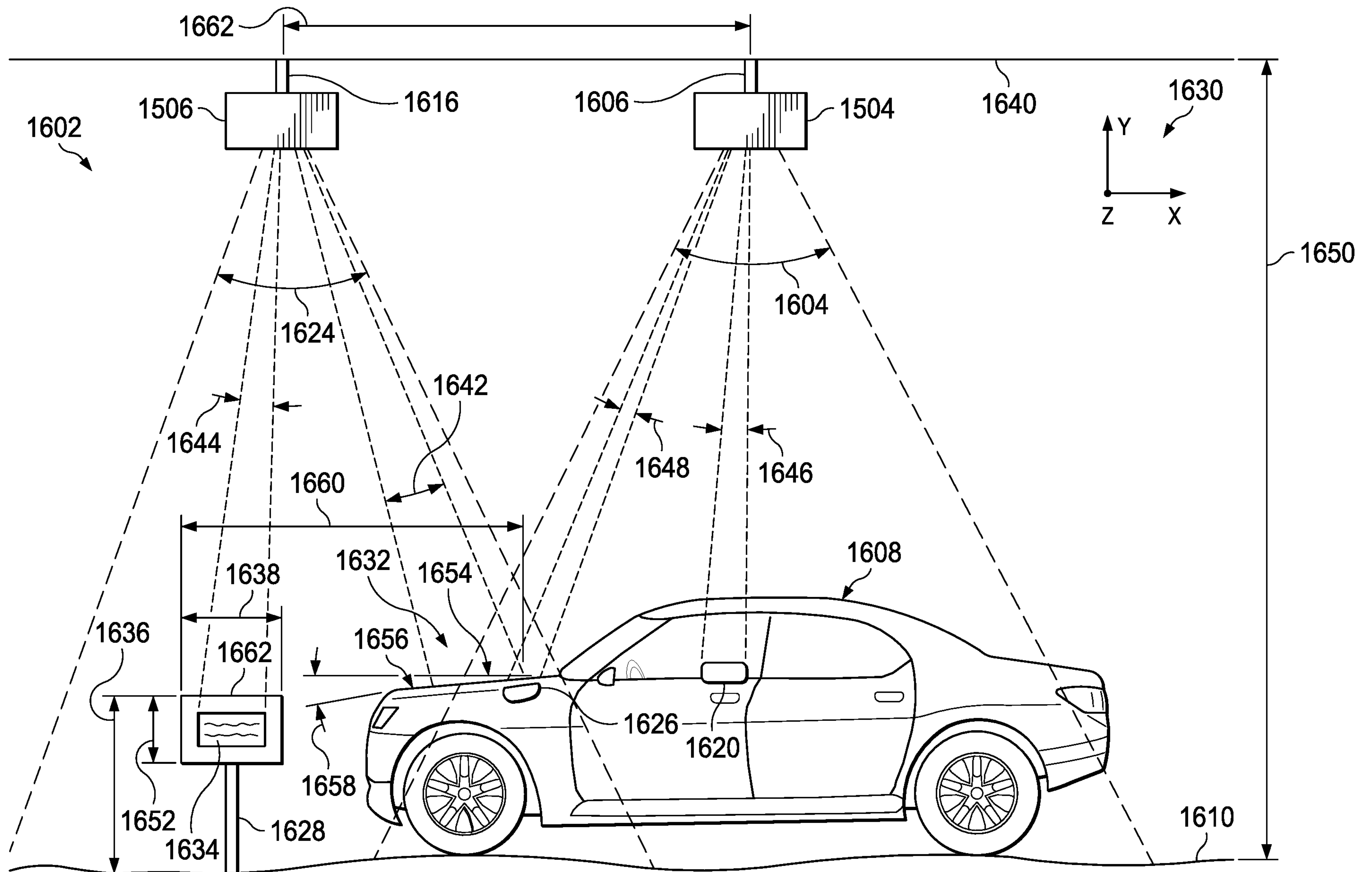


FIG. 16A