



## Public Information Disclosure

Actuality Ref:	ASI-0090
Title	SYSTEM AND METHOD FOR CONVERTING 2-D VIDEO GAMES INTO 3-D VIDEO GAMES
Publication Date	13 November 2005
Inventors	David Aubin (Pelhem, NH), Yigal Banker (Chestnut Hill, MA)
Point of Contact	Gregg Favalora, CTO favalora [at] actuality-systems.com +1 (781) 541-6016 x102
Notes	Patent Pending

## **SYSTEM AND METHOD FOR CONVERTING 2-D VIDEO GAMES INTO 3-D VIDEO GAMES**

### **BACKGROUND**

[0001] Video game developers originally developed video games as two-dimensional (2-D) video games. As video game applications increased in popularity, video game developers subsequently developed three-dimensional (3-D) video game applications having depth features.

[0002] However, because thousands of 2-D video game applications have been developed, there is need for converting the 2-D video game applications into 3-D video game applications to satisfy the gaming demands of video game players accustomed to 3-D video game applications.

### **DESCRIPTION OF EXEMPLARY EMBODIMENTS**

[0003] Referring to Figure 1, before describing a system and a method for converting a 2-D video game application into a 3-D video game application, a brief description of a conventional system 12 for executing a 2-D video game application will be described. The system 12 includes a computer 14 and a display device 16. The display device 16 is configured to display 2-D objects 17 when executing a video game application utilizing the computer 14.

[0004] Referring to Figure 2, a system 20 for converting a 2-D video game application to a 3-D video game application is illustrated. The system 20 includes a computer 22, a display device 24, and a keyboard 26.

[0005] The computer 22 is provided to allow a programmer to modify parameters associated with a 2-D video game application for converting the 2-D video game application into a 3-D video game application utilizing the keyboard 26. The computer 22 is further provided to operably communicate with the display device 24 to render 3-D digital images associated with the 3-D video game application on the display device 24.

[0006] The display device 24 can comprise one of a plurality of different types of 3-D display devices. In particular, the 3-D display device 24 can comprise one of a stereoscopic display device, a multi-planar volumetric display device, a holographic video system, a multi-view 3-D display device, a stereo display device, and a goggle display device. For example, the display device 24 can comprise one or more of the devices described in the following patents and articles that are incorporated by reference herein:

- U.S. Patent No. 2,967,905 entitled "Three dimensional display apparatus";
- U.S. Patent No. 3,140,415 entitled "Three-Dimensional Cathode Ray Tube";
- U.S. Patent No. 6,554,430 entitled "Volumetric three-dimensional display system";
- The article entitled "Volumetric three-dimensional display system with rasterization hardware," authored by G.E. Favolora, R.K. Dorval, D.M. Hall, M.G. Giovinco, J. Napoli in *Stereoscopic Displays and Virtual Reality Systems VIII*, Andrew J. Woods, Mark T. Bolas, John O. Merritt, Stephen A. Benton, Editors, Proceedings of SPIE Vol. 4297, pp. 227-235 (2001);
- U.S. Patent No. 6,100,862 entitled "Multi-planar volumetric display system and method of operation";
- U.S. Patent No. 5,172,251 entitled "Three dimensional display system";
- U.S. Patent No. 6,753,990 entitled "Holographic displays";
- U.S. Patent No. 4,829,365 entitled "Autostereoscopic display with illuminating lines, light valve, and mask."

[0007] A 2-D video game application comprises a software program having the following parameters: (i) 2-D digital images of objects, (ii) 2-D reference locations of

objects, (iii) 2-D vector algorithms for defining movement of objects in two dimensions, (iv) 2-D location boundaries defining a boundary region for movement of the objects, (v) and 2-D vector input algorithms.

[0008] A 3-D video game application comprises a software program having the following parameters: (i) 3-D digital images of objects, (ii) 3-D reference locations of objects, (iii) 3-D vector algorithms for defining movement of objects in three dimensions, (iv) 3-D location boundaries defining a boundary region for movement of the objects, (v) and 3-D vector input algorithms.

[0009] Referring to Figure 3, a method for converting a 2-D video game application into a 3-D video game application will now be described. In particular, the method will add depth functionality to a 2-D video game application to obtain the 3-D video game application. The 3-D video game application can utilize a Cartesian coordinate system or a Quaternion coordinate system for positioning and moving 3-D digital images on a display device.

[0010] At step 100, a programmer or the computer 22 converts the 2-D digital images of objects associated with a video game application to volume rendered 3-D digital images and stores the 3-D digital images in a memory.

[0011] At step 102, the programmer converts 2-D reference locations of the objects to 3-D reference locations and stores the 3-D reference locations in a memory. For example the programmer can convert 2-D reference locations in a (x,y) Cartesian coordinate system to 3-D reference locations in a (x,y,z) Cartesian coordinate system. Alternately, the programmer can convert 2-D reference locations in a (x,y) Cartesian coordinate system to a (x,y,z,w) Quaternion coordinate system.

[0012] At step 104, the programmer converts 2-D vector algorithms associated with the objects to 3-D vector algorithms and stores the 3-D vector algorithms in a memory.

[0013] At step 106, the programmer converts 2-D location boundaries associated with the video game application to 3-D location boundaries and stores the 3-D location boundaries in a memory. For example the programmer can convert 2-D location boundaries in a (x,y) Cartesian coordinate system to 3-D location boundaries in a (x,y,z) Cartesian coordinate system.

[0014] At step 108, the programmer converts 2-D vector inputs algorithms from a user for movement of the objects to 3-D vector input algorithms and stores the 3-D vector input algorithms in a memory.

[0015] At step 110, the computer 22 renders the 3-D digital images utilizing (i) the 3-D reference locations, (ii) the 3-D vector algorithms, (iii) the 3-D location boundaries, and (iv) the 3-D vector input algorithms, having a predetermined perspective on the 3-D display device 24.

[0016] In one embodiment, the foregoing methodology was utilized to convert a 2-D video game application of the game "Asteroids" into a 3-D video game application that can be played on a volumetric display.

### **Original 2-D Asteroids video game application**

1. Images
  - a. 2-D Asteroids
  - b. 2-D Ship
  - c. 2-D Bullets
  - d. 2-D Explosion particles
2. Sound and or music
  - a. Ship fire sound
  - b. Explosion sound
  - c. Background music
3. Game rules (logic)

- a. Shoot by controlling the ship and hitting an asteroid with the bullet. You get points for shooting the asteroids. Once all are shot restart over again.
4. Game physics
  - a. Rotate left or rotate right the ship (positioning)
  - b. Ship thrust (movement)
  - c. Ship firing
  - d. Asteroid hitting bullet
  - e. Asteroid hitting ship
  - f. Draw ship
  - g. Draw asteroid
  - h. Draw bullet
5. Input
  - a. Keyboard left/right arrows and spacebar for shooting
6. Game database
  - a. 2-D Ship position
  - b. 2-D Asteroid position
  - c. 2-D Bullet position
7. Data files
  - a. Ship design in 2-D
  - b. Asteroid design in 2-D
8. View point
  - a. Represented in 3-D coordinates is  $(x,y,0)$ 
    1. Where x is from -.5 to .5 screen width
    2. Where y is from -.5 to .5 screen height

### **Modified 3-D Asteroids video game application**

1. Graphics
  - b. 3-D Asteroids – Added depth (z)
  - c. 2-D Ship – Made it look 3-D with physics
  - d. 3-D Bullets – Made 2-D lines in to 3-D sphere's

- e. 2-D Explosion particles – didn't modify due to polygon count
2. Sound and or music
    - a. Ship fire sound – same
    - b. Explosion sound - same
    - c. Background music - same
  3. Game rules (logic)
    - a. Shoot by controlling the ship and hitting an asteroid with the bullet. You get points for shooting the asteroids. Once all are shot restart over again. - same
  4. Game physics
    - a. Rotate the ship left, right, forward and backward
    - b. Ship thrust (movement) – can move along x,y,z
    - c. Ship firing – can move along x,y,z
    - d. Asteroid hitting bullet – detect along x,y,z
    - e. Asteroid hitting ship - detect along x,y,z
    - f. Draw ship – used 2-D ship and 3-D scaling, and flipping to make it look 3-D, in 3-D coordinate system
    - g. Draw asteroid – draw in 3-D coordinate system
    - h. Draw bullet – draw sphere in 3-D coordinate system
  5. Input
    - a. Keyboard left/right arrows to rotate left and right along x axis
    - b. Keyboard up/down arrows to rotate left and right along z axis
    - c. Keyboard 0 for thrust
    - d. Joystick left/right to rotate left and right along x axis
    - e. Joystick up/down to rotate left and right along z axis
    - f. Joystick button for thrust
    - g. Joystick button for fire
    - h. mouse left/right wheel to rotate left and right along x axis
    - i. mouse up/down wheel to rotate left and right along z axis
    - j. mouse button for thrust
    - k. mouse button for fire

- l. keyboard h for hyperspace
  - m. mouse button for hyperspace
  - n. joystick button for hyperspace
  - o. spacebar for shooting
6. Game database
    - p. Ship position – convert to use x,y,z
    - q. Asteroid position – convert to use x,y,z
    - r. Bullet position – convert to use x,y,z
7. Data files
    - s. Ship design in 2-D
    - t. Asteroid design in 2-D
8. View point
    - a. Represented in 3-D coordinates is (x,y,z)
      1. Where x is from -.5 to .5 screen width
      2. Where y is from -.5 to .5 screen height
      3. Where z is from -.5 to .5 screen depth

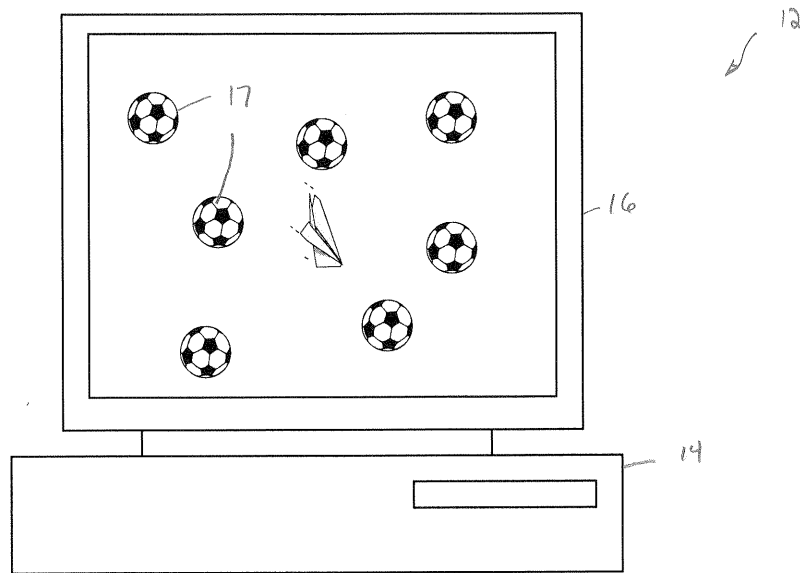


Figure 1

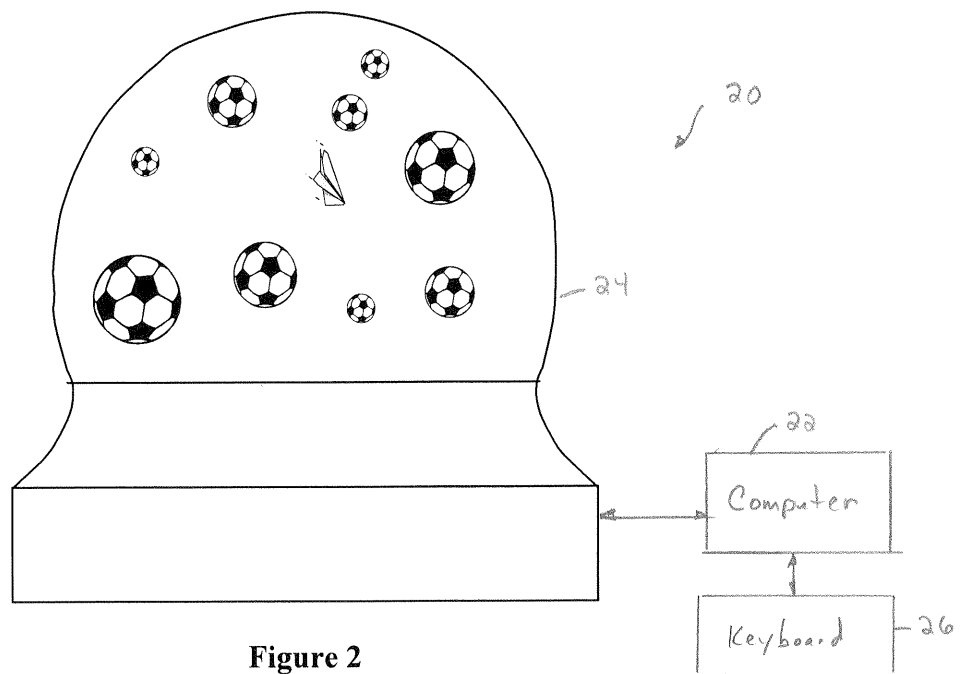


Figure 2

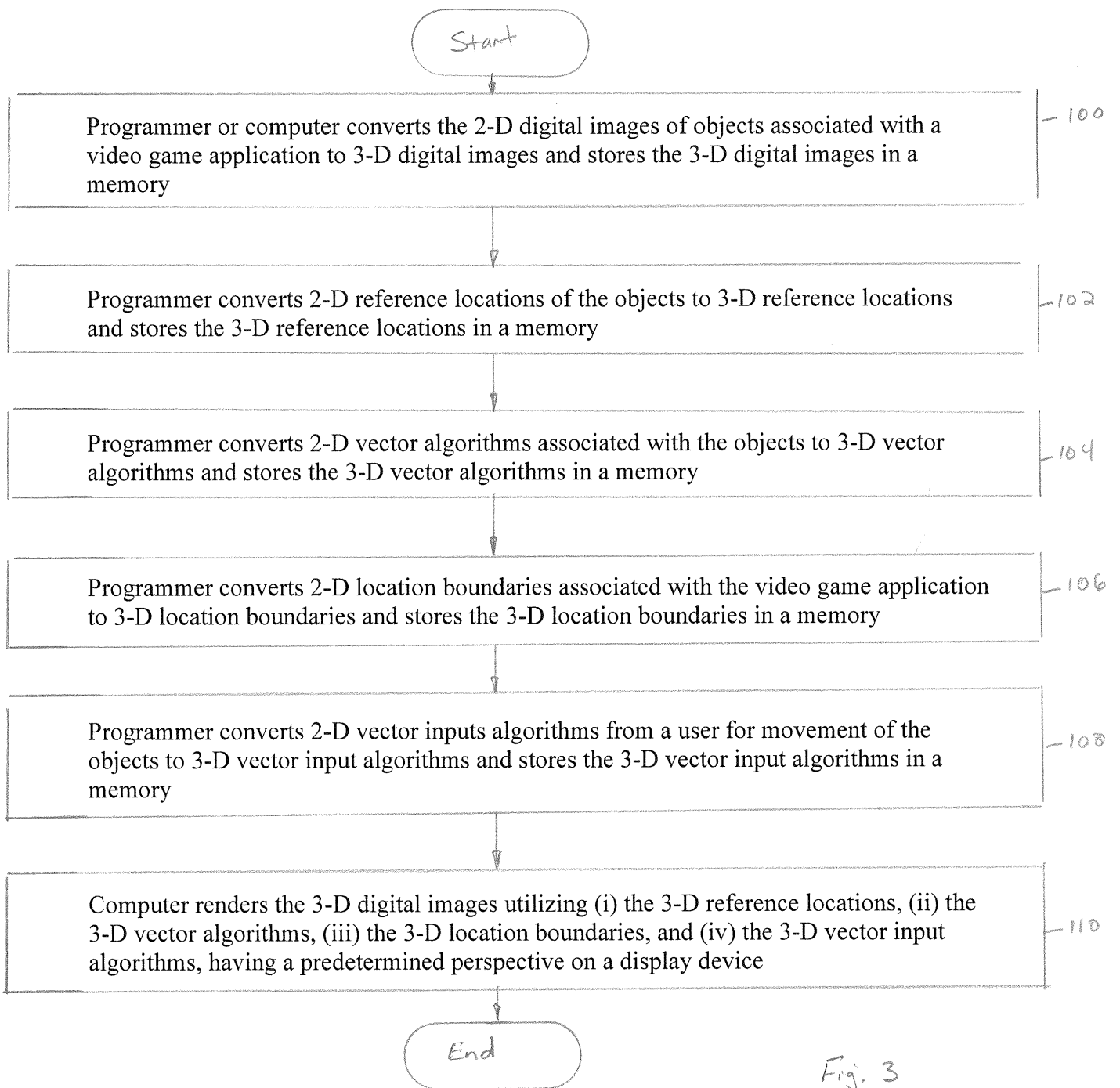


Fig. 3