

Using a Floating Origin to Improve Fidelity and Performance of Large, Distributed Virtual Worlds

Chris Thorne

*School of Computer Science and Software Engineering
The University of Western Australia
chris@ping.com.au*

Abstract

*Large Virtual Worlds (VWs) are increasingly common in the computer graphics areas of simulation, games, geospatial or scientific visualisation. In such VWs, simulated motion of the viewpoint and other objects becomes jittery and lacking in realism when far from the world's origin. Shape and appearance of objects can also degenerate. Since these effects depend on position in space, they will be collectively termed **Spatial Jitter (SJ)**.*

Traditional solutions to SJ embody the idea that viewpoint motion through the VW must involve a change of the viewpoint's coordinates. This notion of viewpoint motion increases design complexity and the processing overhead of code written to counter SJ.

A Floating Origin (FO) approach is presented which floats the world's origin with the viewpoint when navigating a continuous VW, eliminating SJ effects and lowering design and processing overheads. It ensures constant high fidelity in contrast to the continuously varying fidelity of conventional solutions.

1. Introduction

Imagine yourself at the top of a ramp leading down and away into the distance. It seems mirror smooth but as you walk down, small irregularities appear like tiny staircase steps. Further down your progress gets quite bumpy. Eventually your feet can fit comfortably on each step. Later, you have to take small jumps. You conclude that before long the next step will result in a fall and perhaps broken bones!

The diabolical staircase with ever larger steps represents the floating point number line, with its origin at the top and ever increasing gaps between one representable number and the next. On a digital computer, there is a discrete set of representable floating point numbers which *are not distributed evenly* [5]: the gap between one number and the next increases with its size, or distance from the origin. The

gap between 1 and the next number is called the machine epsilon, E_m [7]. The gap between x and the next number is $\sim x \cdot E_m$. In general,

$\text{gap} = P_{2(x)} \cdot E_m$, where $P_{2(x)}$ is the largest power of 2 that is less than $|x|$.

In a VW, the position of objects is defined by coordinates, a notation that denotes distance from origin [6]. Modern computer graphics hardware uses floating point numbers for the coordinates of objects [4,16]. Therefore, like the staircase, the further coordinates are from the origin the larger the gaps between them.

It follows from the above that moving something from one position to another will show increasingly larger jumps with increasing distance from origin, leading to SJ. Figure 1 illustrates the jittery motion of two dots moving far from the origin. SJ effects have been noted before [1, 9, 13, 18, 26, 27, 28, 29].

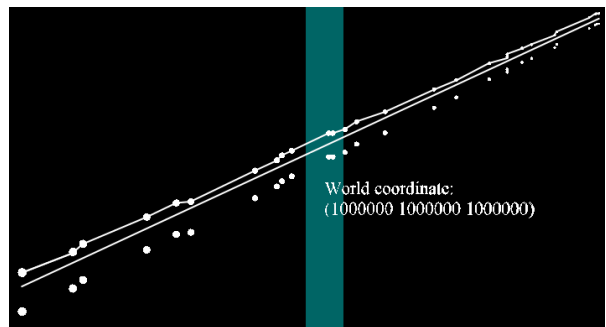


Figure 1. Snapshots of two moving dots overlaid into one image. The dots are near (1000000, 1000000, 1000000) and move from the distant upper right to the near lower left. Each upper dot snapshot is connected by a line, tracing its jittery path. A straight line underneath highlights the uneven motion of the dots.

The total number of representable values a floating point number can have depends on the precision [5] of the floating point representation used. In general, the precision of a real number is the number of decimal

digits in it which are treated as significant for computations [30]. Current graphics hardware has a maximum of 23 bits precision, the precision of the Single Precision Floating Point (SPFP) standard [8], for the coordinates of objects. Often, the precision used can be less [4] and a lower precision means larger gaps between coordinates.

Poor understanding of SJ and its cause, and difficulty grasping the idea of moving the origin instead of the viewpoint, have been barriers to implementing a FO solution. Another barrier has been that existing designs can be difficult to convert to a FO approach. Consequently, this paper aims to remove the enigma surrounding the SJ problem by explaining it clearly and further aims to show how an optimal and complete solution can be attained.

In order to better explain different approaches to preventing SJ, the VW application is divided into a *display system (DS)* and *object system (OS)*. The OS is responsible for defining, tracking and handling the interaction of objects. These objects are given to the DS which generates a user's view into the VW (*rendering*). The DS uses a representation of the objects in a form more suitable to fast rendering and has operators that affect object appearance. Among the operators are *transforms* which shift objects so they are positioned correctly in relation to each other.

The following describes how SJ has been addressed and how some people had difficulties understanding the problem. The FO approach is then described with emphasis on how designers need to change the way they think about SJ and the solution. The general design for a distributed FO system is presented and compared with conventional solutions. The discussion also describes how the FO method should be combined with level of detail streaming and dynamic placement to complete the approach. The benefits of FO are summarised in the concluding remarks.

2. Traditional approaches

Traditional approaches to SJ fall into three classes: on-the-fly shifting of coordinates, subdivision of the world into local coordinate systems and piecewise shifting of world coordinates (WC).

2.1. On-the-fly shifting of coordinates

The on-the-fly approach to jitter shifts objects and the viewpoint close to the origin before calculations are performed that will noticeably affect how they are rendered. The shifting involves changing the object coordinates or transforms that affect their coordinates.

In an article on simulating a solar system, Oneil [13] describes dealing with jitter by shifting visible objects just before they are rendered to each *frame*. A frame is

an image in a series sent to the user's display at a rate of at least 30 frames per second to give the appearance of smooth animation. Oneil's algorithm temporarily sets the viewpoint position at the world origin and subtracts the true viewpoint position from that of all visible objects. Thus everything used in rendering a frame now has small, accurate coordinates and the calculations are consequently higher fidelity, avoiding jitter effects. Once the frame is rendered, viewpoints and object positions are restored to their previous values.

Other on-the-fly examples can be found in online articles and forums on game development sites [26, 27, 28, 29]. For example, in the blitzcoder forum [28], "Andy" proposed a periodic on-the-fly method: every time the viewpoint became, for example, 10,000 from the origin, it was shifted back to the origin and all objects were shifted by the same transform to keep them in correct relative position.

2.2. Multiple local coordinate systems

Many VWs were divided into smaller coordinate regions enabling easier management of detail and minimizing jitter at the same time. This segmentation requires additional structures and management overhead to handle movement between regions.

The Dungeon Siege game segmented its world into *SiegeNodes*, each with its own local coordinate space. When the viewpoint crossed a boundary between nodes, the local coordinate system changed to that of the node being entered and a "Space Walk" began [1]. The space walk visited each active node and recalculated coordinate transforms to shift objects closer to the new local origin. This ensured coordinates did not get large enough to cause jitter.

The connection between nodes in Dungeon Siege was similar to what has been called *portals* in other segmented VWs. Eternal Lands (EL), a Massively Multiplayer Online Game (MMPOG), used portals which teleported the player from one segment to another. EL had one world map divided into continent maps. It also used ships as a natural transporter system. Other games also made use of natural transporter systems to move between segments [12, 15, 31].

In the Morrowind game [12], a player walking far enough along a path would experience a halt where movement was temporarily blocked, while a message like "loading external environment" was displayed. This hiatus marked the transition from one segment of the game world and the next.

Flight simulators need to counter jitter because they cover very large areas. FlightGear, for example, divided the earth into tiles, each with a local coordinate system. It made aircraft and other object coordinates relative to the reference point of the tile they were in. "This reference point (for purposes of avoiding floating

point precision problems) defines the origin of a local coordinate system” [14].

Another simulation system using local coordinate spaces was Spline, used by Barrus et al. [3] to build the Diamond Park virtual world. Spline supported the subdivision of the larger world into smaller *locales* and the efficient communications between them. When an object moved from one locale to the next its coordinates were transformed to be relative to the new locale origin.

VGIS, a GIS application for modeling the earth, subdivided the world into regions. VGIS was described in a paper by Lindstrom et al. [10] who thought the subdivision was necessary: “This is necessary as the precision afforded by current graphics hardware is insufficient for representing detailed geometry distributed over a large volume in a single coordinate system”.

2.3. Piecewise shifting of coordinates in a continuous virtual world

Dungeon Siege appears like a continuous world because there is no noticeable hiatus while moving between segments, but it is not a true continuous world because it subdivided the world into SeigeNodes. A true continuous world has a single world coordinate system with no artificial segmentation of that space. Examples of applications built on a continuous world space are Terravision [9], planet-earth [24] and applications using the GeoVRML [18] technology. These applications perform a shift when a special type of viewpoint is selected.

GeoVRML is a geospatial extension to the VRML [25] language. In GeoVRML, whenever one moves to a new area of interest, a GeoViewpoint is used. A GeoViewpoint contains the origin of the area of interest and the position of a nearby viewpoint, both in WC. The GeoViewpoint reverse transforms the world by subtracting the origin from the WC of the nearby viewpoint and other objects resulting in small coordinate values, thus avoiding jitter. In this way, the world is shifted in a piecewise fashion as the user goes from viewpoint to viewpoint.

Apart from viewpoint selection, another way to move through a virtual world is by free navigation. Free navigation is when the user operates controls for walking, flying or other continuous movement. In GeoVRML, there is nothing to stop one from freely navigating to a point where SJ occurs. Terravision is based on GeoVRML and thus has the same properties, except the free navigation issue is managed by periodically re-translating the world so the viewpoint is at the origin, and everything else is closer to it.

The planet-earth project [24] uses a continuous shifting of the world. This is a combination of

piecewise shifting via viewpoints like GeoVRML and TerraVision plus continuous reverse translation of the world during free navigation. The latter is what distinguishes the FO method from other continuous world methods like GeoVRML and TerraVision and is described in the next section. It also distinguishes FO from the traditional SJ solutions which have the viewpoint always moving relative to the origin.

3. Floating origin approach

3.1. Reversal of navigation thinking

In the review of traditional systems that manage jitter, there is a common thought process apparent in their design. The apparent assumption is that while freely navigating or moving from one viewpoint to another, the coordinate of the viewpoint must move through the virtual world: i.e. its value must change. Coordinate shifting is only added on just prior to rendering frames or to periodically reduce the viewpoint and object coordinate values. Even with GeoVRML and TerraVision, free navigation moves the viewpoint through the world space.

The FO approach is a departure from such conventional navigation thinking. As shown in Figure 2, instead of allowing the viewpoint to move in the world, the world is reverse transformed to position the desired point at the origin. The origin is not fixed relative to the rest of the world but floats with the viewpoint. Therefore, the viewpoint is always centered at the highest fidelity place: the origin, even when freely navigating. The effect of using a FO is that there will be no observable SJ. Since the origin is the world's center, this type of viewpoint is termed here a *centered viewpoint*.

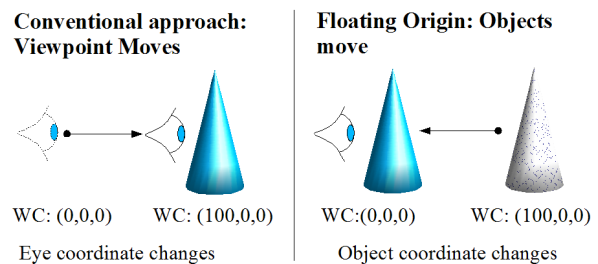


Figure 2. Comparing conventional and floating origin navigation.

3.2. Understanding the problem, grasping the solution and implementing it

Comprehensive searches revealed the closest approaches to FO that have been formally documented are the GeoVRML and Terravision approaches already

described [9, 18]. They are in the narrow field of web based geospatial 3D. There were also a small number of online articles and forum topics [13, 26, 27, 28, 29]. This section looks at how people came to understand SJ and their approaches to it.

In the informal online documentation, there is evidence people have had difficulty understanding the problems caused by large distances from the origin. On the blitzcoder community forum [26], “Nukomod” initially did not understand what floating point coordinates and distance from origin would do to motion and rendering: “The further a mesh travels from the 0,0,0 origin the more it begins to 'break apart'. There seems to be some inaccuracy in positioning child entities and the whole model begins to 'shake'.”

In the BlitzBasic forum, “Second Chance” [29] had a problem with disappearing planets in a space game: “...Then it's assigned its orbital distance from the star (0,0,0). Everything looks good until you get out to about the orbit of Uranus (29,000,000 game units) and beyond, at that point the planets start to wink in and out of existence.”

In a gamedev.net forum, “Lode” [27] was surprised by a problem with rendering planets in a space game: “... just to test I made some cube-planets, and even if they're only 1000 units they're uglier than the EXACT same scene 1000 times smaller! ... for bigger scenes with more cubes at larger distances, it looks even worse than this”.

In another blitzcoder forum topic, “Nmuta” [28] had similar problems with objects 10,000 units along the z axis: “Anyone else hear about this problem occurring if you don't normalizing your world regularly, and getting graphics problems when you go too far away from 0,0,0 ? I need to know what that means.”

The above quotes show that people working on large scale worlds can be surprised when encountering the limits of floating point coordinates and are initially at a loss as to what to do about it.

There is even evidence that once people understood the jitter problem and a general solution: that viewpoint and object coordinates should be closer to the origin, they still had difficulty grasping the FO concept. For example, on the blitzcoder forum [28], Nmuta says: “It's just weird, the thought of it. Instead of moving a person, you are moving several tons of land and buildings and enemies!!!! ... The whole concept of 3D in general is such an interesting illusion”. “Mt Dew” says: “When first suggested to me that I needed to change my system from moving my person through the solar system to moving the solar system around the person ... well ... i was a bit daunted”. “Andy” stated: “This is probably the single most difficult concept for new programmers to understand, but it is vital to the process.”

Leclerc et al. and Reddy et al. [9,18] understood SJ and created the piecewise continuous world solutions

of GeoVRML and TerraVision. However, they did not take the method to a full FO solution. Since these authors published in a fairly narrow field, this might explain why their understanding did not disseminate to the broader community and reach the games and flight simulation audience.

The lack of adequate, widespread literature on the subject and evidence of difficulty thinking of a solution in terms of continuous reverse transformation are the main motivations for writing this paper. The following section describes the design for FO navigation.

3.3. Design for floating origin navigation and rendering

To implement FO navigation, the design of the client needs to support reverse transformation of the world whenever the viewpoint is set to a WC and during free navigation. This section describes a general distributed architecture to meet these requirements.

As shown in Figure 3, reverse transforming the world can be achieved by placing a top level transform, the *world transform (WT)*, over the entire set of objects. Whenever a viewpoint is selected, the inverse of the viewpoint's coordinate is applied to the WT. The result is that the objects are shifted in reverse towards the viewer who stays at the origin.

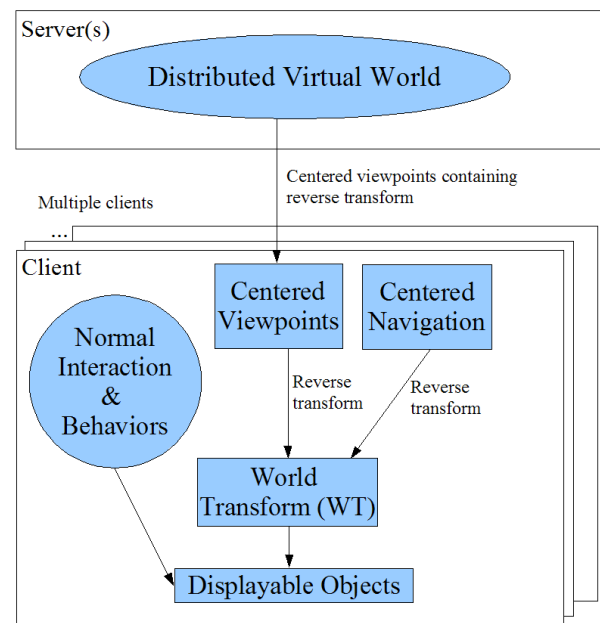


Figure 3. Structure of the client, showing how centered viewpoint and navigation components affect the WT which, in turn, transforms objects. Note how other navigation and behaviours do not affect the WT but act directly on the objects.

Free navigation controls also apply a reverse transformation to the WT. Other interaction and behaviors, such as moving an object, opening a door, or pressing a button, all happen in the normal fashion and do not affect the WT.

A necessary consequence of the WT is that viewpoints must not be under the WT itself because this would form an endless feedback loop with the WT. The design therefore separates viewpoints from the objects and object transforms that are under the WT.

Normally, an application must keep track of where objects are in relation to each other and to the user's viewpoint in order to manage relationships and interactions between objects and render things correctly. However, in the FO approach, the object coordinates in each user's DS are unique to that user's position in the world because they are always the smallest values relative to the user's current world position. This is one reason why the OS part of the application must be separate from the DS, so it can keep track of viewpoint and object position in WC while the DS separately transforms the local coordinate system of each user. This also implies the application has to keep track of a unique WT for each user.

A VW must be distributed to support multiple participants, such as in a MMOG, distributed simulation [2,11] or for collaboration. The OS maintains the overall VW picture and oversees interaction between participants, so it is the OS which needs to be distributed, usually by putting part of it on one or more servers. Similarly, distributing the FO method is a matter of separating the OS into client and server parts and defining a network communication protocol to transmit the necessary information between the parts (Figure 4).

It is the responsibility of each client DS to provide position updates as the user navigates so the OS can track positions in WC. In a distributed application, the client's DS will pass its updated user position to its local OS which now sends it on to the server OS. This is similar to Massively Multiplayer Online Games (MMOGs): the online servers must get user position updates streamed back constantly in order to correctly perform collision detection, physics and interactions and update the visible picture for all users. The main difference is that the server OS is keeping track of reverse transforms for each user locale in addition to tracking object users' WC positions.

Apart from changing the navigation code to reverse transform the WT, the collision detection algorithm may also need modifying. In a conventional system, collision detection activates when navigation causes a volume around the viewpoint to intersect with an object. With FO, because the viewpoint does not move, collision detection has to activate when an object is moved to collide with the volume around the viewpoint.

In a VW, the user only needs to see the part of the world in the vicinity of the current location. Therefore, only detail from this part, the user's locale, need be downloaded from server to client. That is no different from MMOGs which already do similar management of the amount of information the client system has to handle and display. This management technique is called Level of Detail (LOD) and not only limits the information displayed but what has to be transformed.

In the FO approach, LOD would be streamed from the server on demand to keep the overhead of changing detail to a minimum as the user moves. In addition, whenever a user selects a viewpoint the server will pass the client the appropriate reverse transform to shift the local DS's objects to the center of the locale. This information may be in the form of a centered viewpoint (Figure 2) or raw transform information.

Many objects such as terrain and other static things do not have to exist in the DS until they are about to become in visible range. Then they are placed dynamically in the environment. This just in time dynamic placement means objects are positioned with optimum accuracy because their DS coordinates are minimised to the smallest they can be before being placed.

To transmit LOD, position and other information between client and server, current distributed VW applications commonly use TCP/IP sockets [20], or UDP [21] for communication between client and server. Some systems use multicast [22] and others use HTTP [19]. A combination of two of these methods is also possible. The planet-earth project [23] uses HTTP because it is widely supported and the fact that it requires no setup effort for the user.

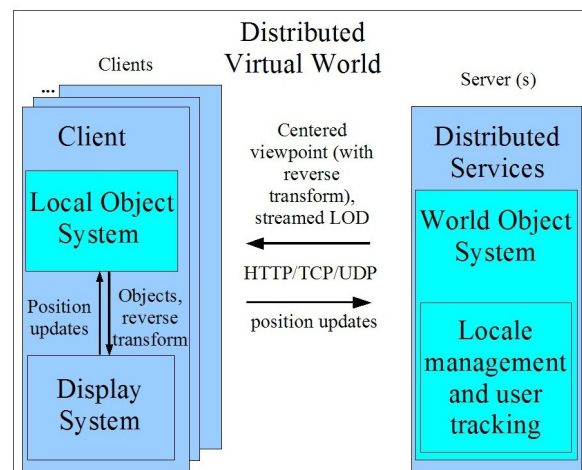


Figure 4. Architecture of the distributed VW application showing separation of client and server parts of the OS and how transformations and position streaming is communicated.

Figure 4 shows the logical architecture of a distributed VW application with the OS divided into client and server components. Locale, LOD, transform and position information is exchanged between the server and client parts of the distributed OS. The client OS applies reverse transforms to the visible world via the DS, giving the user the impression of moving through the world.

From this section, the following design aspects are required to implement FO:

1. a top level transform, the WT, that affects all the objects in the VW,
2. special centered viewpoints which reverse transform the world through the WT,
3. a free navigation system that applies a reverse transform to the WT,
4. separation of viewpoint transforms from object structures in the DS,
5. distributing the OS between client and server and providing a supporting protocol,
6. modified collision detection and terrain following,
7. streamed LOD, and
8. dynamic, just-in-time placement

4. Comparison of floating origin with traditional methods

4.1 Design complexity and performance costs

There is a performance and design cost to traditional solutions. Morrowind [12], for example, has a large world divided into smaller parts. A player walking far enough along a path will experience the hiatus described in section 2. The delay is caused by the game engine loading terrain and objects for a connecting segment because the player has crossed one of the invisible boundaries that separates segments. Apart from causing a disruption to the immersive experience, such events occupy the system resources enough to make gameplay hesitate for a noticeable time.

Even though Dungeon Siege has no loading screens, its space walk has significant performance costs. It requires a *scaffolding*: a structure of linked nodes that enable the walking algorithm to work its way out from the new center node to other neighbors. It uses considerable processing resources and the frequency of performing recalculations has to be limited: “as infrequently as possible to avoid bogging down the CPU” [1].

Algorithms like Oneil's also involve extra work to modify the transformation for each object. Like other graphics systems, Oneil's would require a structure through which objects can be located in relation to each other and the world. To transform each object the structure has to be traversed to access each object's

transform. The work for transform calculations plus scaffolding traversal is proportional to the number of objects. Oneil's algorithm runs every frame, so, for a large number of objects, would represent a large processing overhead at a good frame rate (greater than 30 times a second).

In the blitzcoder forum [28], Andy said that with his method, shifting all objects in one go is too computationally expensive and would produce a noticeable delay, so the algorithm was enhanced to progressively shift objects, a small number at a time, while allowing other operations to continue.

In contrast, the FO method does not need the scaffolding and transform operations used by Dungeon Siege, Oneil and others on every object, as the objects are automatically translated via the WT. Therefore, a lot of processing overhead is removed.

For viewpoint movement the reverse transformation requires no additional code above what is normally there because the navigation code that previously transformed the viewpoint now (inversely) modifies the WT instead. It is just the same operations applied in reverse on a different place. Therefore, there is no additional performance cost to using FO. This is the same conclusion that Nmuta came to: Nmuta [28]: “Yet in terms of rendering, the computer doesn't really know the difference I suppose. It would still be the same number of polygons moving just as fast in the same direction.”

Although GeoVRML does not have the overhead of transitions between segments because it uses a continuous world, free navigation still allows the viewpoint to move far from the origin where jitter can occur and there is no automated control in place to prevent it. There is also nothing stopping the GeoViewpoint position being set to the other side of the world to its origin, in which case jitter would be very noticeable. The FO method does not separate viewpoint position from the origin, thus eliminating these potential problems.

4.2 Coordinate systems and representation

The FO method exploits the high fidelity region around the origin of the floating point space used by modern graphics systems. If other than floating point was used, this would not be possible. Floating point with a precision greater than 8 bits only came into wide use in graphics hardware from 2002, when the Pixel Shader 2.0 specification was released [16]. Prior to that, fixed point and integer numbers were commonly used for coordinates and they have an even distribution of representable numbers. Therefore, a FO approach would not have benefited visualisation. This background may also have contributed to the difficulties some people had in understanding SJ and its

FO based solution on modern hardware.

There is no need for representing the coordinates of objects and viewpoint in double precision floating point as Oneil thought was necessary [13] because the dense precision region around the SPFP origin gives sufficient accuracy for smooth motion and accurate rendering in VWs up to and larger than the earth [24]. Not having to use double precision for operations also reduces processing overhead because SPFP operations are faster than double precision operations [13].

Multiple local coordinate systems like in EL, FlightGear and VGIS are also not required in an FO approach. All supporting structures and algorithms to handle transitions between segments, as in Morrowind, Eternal Lands and Dungeon Siege are no longer needed.

4.3 Applying a FO design

Despite its efficiency benefits, the FO method cannot be fully exploited if the OS and DS are not designed to support it. In online forums, there is evidence that people attempted an approach similar to FO but found it was not efficient to modify an existing application. For example, Nmuta [28] tried using an FO type of system to stop SJ with his space game, but gave up and went with a an on-the-fly method when he found the overhead of implementing FO on top of an the B3D engine was too high: “keeping the player at 0,0,0 all the time as MSW suggested is proving to be a little too tricky to implement given my current engine's set up, so I think I might use your way, with the grid.”

In the same forum “Andy” described the cost of implementing FO on B3D, saying “You would end up having to write code to do everything that B3D does already and add overhead.” He further said that the progressive piecewise approach he proposed was only a few lines of code but modifying B3D's “movement, rotation and collision code will add hundreds of lines and slow the program”.

From the above one can conclude that for a VW to make best use of the efficiency of FO it would have to be designed from the beginning with that method in mind or already have a design that can be easily adapted.

As all of the conventional approaches to SJ allow the viewpoint to move relative to the origin, *accuracy of motion and rendering will continually vary*: i.e. the fidelity will be unpredictable. Many developers may not realise the cause when the quality of rendering is subtly wrong when viewed from a large coordinate. Small seams may appear in one place but not another. Small unaccounted for inaccuracies could plague developers who do not know they are simply caused by SJ. FO ensures rendering is always performed from a centered viewpoint and therefore ensures all motion

and rendering has the same quality.

The absence of complex scaffolding and code to handle transitions from one local coordinate system to the next results in a simpler design. As pointed out in section 4.1, this simpler design has less processing and code overhead, as long as the implementation is not burdened by an existing code base with incompatible design.

5. Conclusion

One may sympathise with parents who complain the younger generation are lazy: they expect everything to come to them. In a virtual world the rules of physics do not apply, it is just as easy to bring the world to you as it is to move yourself in the world. This paper has shown that for modern graphics hardware based on floating point numbers, it is better to keep the viewpoint at the origin and reverse transform the world. So in terms of virtual worlds, the young have the right way of thinking.

The problem of spatial jitter and several classes of conventional approaches to the problem have been described. The benefits of the floating origin solution are that it:

1. eliminates observable spatial jitter,
2. lowers design complexity,
3. lowers processing overhead,
4. provides constant, optimum fidelity of motion and rendering compared to the continually variable fidelity of conventional approaches, and
5. allows end user devices to operate within their precision limits without spatial jitter problems.

These benefits have to be weighed against the need for a new design that will support floating origin navigation efficiently.

Floating point suits the view-centric nature of computer graphics: the local region around the viewpoint is what the user sees and that is where most of the accuracy of rendering and motion needs to be. A floating origin allows this natural benefit of floating point to be applied to the rendering of every frame. When combined with level of detail streaming and dynamic placement a floating origin optimizes the quality of motion, interaction and rendering throughout large, continuous virtual worlds by minimizing spatial jitter.

6. References

- [1] Bilas, S. 2003. The Continuous World of Dungeon Siege, Gas Powered Games, www.drizzle.com/~scottb/gdc/continuous-world.htm.
- [2] Blais, C., Brutzman, D., Horner, D., Niclaus, S. 2001. Web-based 3D technology for scenario authoring and visualisation: the SAVAGE project, *Interservice/Industry*

Training, Simulation and Education Conference (IITSEC) 2001, Orlando, Florida.

[3] Barrus, J. W., Waters, R. C., and Anderson, D. B. 1996. Locales and Beacons: Efficient and Precise Support For Large Multi-User Virtual Environments. Retrieved: March 1, 2003, from <http://www.merl.com/papers/TR95-16a/>.

[4] Carmack, J. 2003. Carmack's .plan files blog, 29/1/03, <http://doom-ed.com/blog/category/doom-ed/john-carmack/>.

[5] Computer Architecture, section 2.6, <http://csep1.phy.ornl.gov/ca/node8.html#SECTION00026000000000000000>.

[6] Definition of Coordinate, <http://www.chatham.k12.ma.us/techpage/dictionary/a.htm>.

[7] Horvath, G. and Verhoeff, T. 2003. Numerical Difficulties in Pre-University Informatics Education and Competitions, *Informatics in Education*, 2003, Col. 2, No. 1, 21-38.

[8] IEEE 754-1985 (R1990). *IEEE Standard for Binary Floating-Point Arithmetic*, <http://standards.ieee.org/>.

[9] Leclerc, Y., Reddy, M., Eriksen, M., Brecht, J. and Colleen, D. 2002, *SRI's Digital Earth Project*, Technical Note No. 560, Artificial Intelligence Center, SRI International, Menlo Park, CA.

[10] Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Bosch, A., and Faust, N. 1997. An Integrated Global GIS and Visual Simulation System. Tech. Rep. GIT-GVU-97-07, Georgia Institute of Technology, Mar. 1997.

[11] Macedonia, M., Zyda, M., Pratt, D., Barham, P. and Zestwitz, S., (1994). "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments", *Presence: Teleoperators and Virtual Environments*, Vol. 3, N. 4, 1994, pp.265-287.

[12] Morrowind game, Bethesda Softworks, <http://www.elderscrolls.com>.

[13] O'Neil, S. 2002. A Real-Time Procedural Universe, Part Three: Matters of Scale, *Gamasutra*, July 12, 2002.

[14] Olson, C.L. 1997, FlightGear Scenery Generation Tools, <http://www.jp.flightgear.org/Docs/Scenery/SceneryGeneration/SceneryGeneration.html>.

[15] Pirates of the Caribbean game, Bethesda Softworks, <http://pirates.bethsoft.com>.

[16] Pixel Shader 2.0 precision, <http://www.digit-life.com/articles2/ps-precision>.

[17] Privantu, R. 2004. Eternal lands, post mortem, <http://www.devmaster.net/articles/mmorpg-postmortem/part1.php>.

[18] Reddy, M., Iverson, L., Leclerc, Y. and Heller, A.

2001. GeoVRML: Open Web-based 3D Cartography. In *Proceedings of the International Cartographic Conference (ICC2001)*, Beijing, 6-10 August 2001.

[19] RFC 1945 - Hypertext Transfer Protocol – HTTP/1.0, May 1996.

[20] RFC 793 - Transmission Control Protocol, DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION, 1981.

[21] RFC 768 - User Datagram Protocol, J. Postel, ISI, 28, 1980.

[22] RFC 966 - Host groups: A multicast extension to the Internet Protocol, Network Working Group, S. E. Deering, D. R. Cheriton, Stanford University, December 1985.

[23] Thorne, C. 2005. Exploiting an Evolutionary Accident in Web3D Communications to Integrate Application, *Proceedings of the SIGGRAPH 2005 conference on Web graphics: in conjunction with ACM SIGGRAPH 2005: Web Graphics, Session: 3D*, Conference CD ROM, Los Angeles, CA, 31 July-4 August, 2005.

[24] Thorne, C. and Weiley, V. 2003. Earth's Avatar: The Web Augmented Virtual Earth (WAVE), *SIGGRAPH 2003 Conference on Web Graphics*, Session: Visualisation, San Diego, CA, July 27-31.

[25] VRML. ISO/IEC 14772-1:1997. The Virtual Reality Modeling Language, Part 1.

[26] 2004 blitzCoder community forum, Topic Large playing area problems, initiated by "Nukomod" [n.http://64.233.179.104/search?q=cache:3gSbeJsSYEcJ:www.blitzcoder.com/cgi-bin/ubb-cgi/ubbmisc.cgi%3Faction%3Dfindthread%26forum%3DForum14%26number%3D14%26thisthread%3D001610%26go%3Dnewer](http://64.233.179.104/search?q=cache:3gSbeJsSYEcJ:www.blitzcoder.com/cgi-bin/ubb-cgi/ubbmisc.cgi%3Faction%3Dfindthread%26forum%3DForum14%26number%3D14%26thisthread%3D001610%26go%3Dnewer).

[27] 2002 gamedev community forum, Topic Large object in OpenGL look ugly???, initiated by "Lode": http://www.gamedev.net/community/forums/topic.asp?topic_id=106633.

[28] 2003 blitzCoder community forum, Topic: Huge worlds, initiated by "Nmuta" <http://www.blitzcoder.com/cgi-bin/ubb-cgi/postdisplay.cgi?forum=Forum4&topic=000352>.

[29] 2004. BlitzBasic community forum, Topic: Mystery of the disappearing planets, initiated by "Second Chance" <http://www.blitzbasic.co.nz/Community/posts.php?topic=23366>.

[30] Wolfram Research, Mathematica documentation, <http://documents.wolfram.com/v5/TheMathematicaBook/AdvancedMathematicsInMathematica/Numbers/3.1.4.html>.

[31] World of Warcraft web site, <http://worldofwarcraft.com>.