A Distributed Approach for Automatic Speed Adjustment during Navigation in 3D Multiscale Virtual Environments

Henrique Taunay
Tecgraf Institute/PUC-Rio
PUC-Rio
Rio de Janeiro, Brazil
htaunay@tecgraf.puc-rio.br

Daniel Medeiros
Computational Media Innovation Centre
Victoria University of Wellington
Wellington, New Zealand

INESC-ID/Lisboa
Lisboa, Portugal
daniel.medeiros@vuw.ac.nz

Alberto Raposo
Tecgraf Institute/PUC-Rio
PUC-Rio
Rio de Janeiro, Brazil
abraposo@tecgraf.puc-rio.br

Abstract—**Multiscale virtual environments (MSVEs) are virtual environments that encapsulate elements with different scales within a same shared space. They may contain elements with extremely different levels of scale in the same environment and can be found in geographical maps and engineering virtual environments (VEs). Due to its complexity and diverging levels of scale, this type of environment could be accessed using its hierarchical structure to navigate between the levels of scale. But, in geographical maps and engineering VEs users may need to freely navigate through the levels of scale seamlessly to provide improved spatial knowledge. Common approaches to navigate within these 3D virtual environments are based on the Automatic Speed Adjustment approach, where the scene is pre-processed in a data structure and then accessed in real-time to determine optimal speed. A common trend is to migrate this process from the CPU to GPU, but the works that use this approach are still limited to static and/or smaller virtual environments. As the scene grows in complexity, the computational power needed to render the 3D scene and determine optimal speed may be too costly. We propose RMNS (Remote Multiscale Network System) to solve this problem by using a service-oriented approach to compute speed and is asynchronously accessed to determine optimal speed. Our results show that our approach enables the use of dynamic scenes, as the rendering and speed adjustment are decoupled. Finally, the asynchronous nature of our approach showed that the response time is enough to support automatic speed adjustment, while maintaining the rendering in real-time.**

Index Terms—**Computer Graphics, Methodology and Techniques, Interaction techniques, Three-Dimensional Graphics, Realism, Virtual Reality**

I. Introduction

The complexity of some scenes such as geographical and engineering VEs brings about the need to navigate on elements of the scene in different scales. These types of scenes are called Multiscale Virtual Environments, and could contain objects in extremely varying levels of scale [1]–[4]. Such scale variations make it difficult to navigate these environments. Indeed different scales require repetitive and unintuitive adjustments in either velocity or scale, depending on which objects are closer to the observer, to achieve a comfortable and stable navigation. Examples of these types of environments can be found in different areas of expertise such as Medicine [5]–[8], Geography [9] and Engineering [10], [11]. In some cases, the use of multiscale techniques can also benefit some applications not inherently multiscale to facilitate manipulation of objects and navigation throughout the virtual model [12]–[14], to overcome physical constraints of the physical environment and to support collaboration in virtual environments [15], [16].

A form of tackling the navigation is by using Level of Scale (LoS) solutions. In this family of techniques the user may manually control their user representation and VE size [9], use visual landmarks or use hierarchical structures to nest each of the elements of the same level of scale and discretely navigate in each of them [17]. Another way to address the multiscale navigation problem is by using automatic speed adjustment techniques (ASATs) [11], [18], [19], which use the closest geometry position as input to heuristics that determine the optimal navigation speed at a given moment. This sort of method is best suited for VEs based on geographic information, as the user needs to freely explore the environment in order to obtain better spatial understanding. A classic example of this approach is the Cubemap [20], which uses 6 rendering passes to calculate the distance between the camera and the environment, each in a different direction to cover all the environment and determine optimal speed of navigation.

As the virtual environment increases in size and complexity, the computational power needed to calculate 6 rendering passes would be too high, and in these cases the use of such techniques may become impractical. A recent overcome to this problem is by using spatial structures such as Kd-trees, which are commonly used to accelerate rendering of objects in multiple scales [21], [22]. Taunay et al. [19] used a similar structure to pre-process and access the scene in the GPU to determine the movement speed in real time. However, this approach is still not viable for use in dynamic scenes, where the objects are in motion. As the optimal speed calculation is shifted from the GPU to the CPU, it is possible to use a distributed approach to overcome this issue.

In this paper we propose RMNS (Remote Multiscale Network System), in which we use a distributed remote system to tackle the multiscale navigation problem. This system decouples the rendering and optimal speed calculation, where the rendering can be done at the user's computer, which creates a spatial structure at the remote server, following an approach similar to the work by Taunay et al. [19], which is accessed asynchronously to determine optimal speed. Our results show that the use of RMNS enables real-time rendering and quick response times, which enables its use for complex dynamic multiscale scenes.

In the following sections we clarify the rationale for the multiscale navigation problem used in our system design, accompanied by a comprehensive description and a performance test, followed by findings and possible instructions for future work.

II. Related Work

The main classification of multiscale environments differs on how the system handles the transitions between the different levels of scale. On the Level of Scale (LoS) type the user navigates through the different levels of scale by scaling himself or the environment; on Automatic Speed Adjustment Techniques (ASATs), the transition is made by adjusting automatically the speed between the levels of scale for a seamless navigation.

The different levels of scales of these environments present themselves with elements that can be represented by an hierarchical structure, where each part of the structure is a context of objects in the same level of scale. Kopper et al. [6] uses the concept of visible landmarks [13] to navigate through the hierarchical structure of a virtual representation of the human body. They also concluded that automatic scaling is more efficient in comparison with manual scaling when navigating through the visible landmarks. Bacim et al. [5], on the other hand uses a tree-like structure to navigate through the different levels of scale of the human body. In other cases where the environment represents spatial data, such as maps and oil fields, it is important for the user to navigate freely on the environment to gain better spatial knowledge. But freely navigating in a 3D virtual environment can prove to be problematic, even for the most experienced users [23], and possibly deal-breaking for laymen, specially when dealing with massive multiscale scenes. On this type of environments the system should be responsible for determining the optimal speed according to the level of scale that the user is currently in.

The speed of navigation on ASATs-like environments is based on the complexity of the model. In some cases the user restricts its trajectory by establishing a point of interest (POI) [24] and navigating towards it. Argelaguet et al. [25] propose a perceptual-based approach based on visual attention models, which is based on optimizing navigation speed at each point of a camera path according to the desired perceived motion. Another example of this type of work is the Drag 'n Go technique, where the user utilizes multi-touch gestures to navigate to a POI with logarithmic speed [26]. But in some cases the use of linear speed to navigate in virtual environments is not efficient because of the high complexity and multiscale nature of some environments. On oil fields (a type of engineering scenario) for example, there are elements varying in a scale of $1{:}10^7$ from the smallest object (an oil tube with a 15cm radius) to the largest one (a seismic object with possibly kms of extension in all three dimensions). In such complex scenes there is normally the need of using methods to analyse the virtual scene to determine the best speed on a given level of scale. While Freitag et al. [27] use the viewpoint quality, which is the amount of relevant of information on the view frustum, other works use the Cubemap [11], [20], [28], which consists on storing 6 render passes to choose the optimal velocity by fetching the nearest point to the user at a given frame.

Another approach to tackling the 6DOF multiscale problem was the adaptive navigation technique by Argelaguet and Morgan [18], where not only the observers navigation speed is automatically adjusted, but the camera's rotation speed as well, in order to reduce jerkiness during the interaction. Argelaguet and Morgan, instead of taking in as input the nearest object, worked with a combination of: a Time to Collision map, a grid where each cell represents the time to reach each rendered pixel given the user's current speed; and an Optical Flow map, a grid where each cell represents the amount of displacement (in screen space) between two consecutive frames. Both maps when applied to a custom algorithm can provide - what the author describes - as the perceived user's speed, which when compared to a hard coded optimal perceived speed (by configuration) serves as a reference for maximum translation and rotation variations between frames.

However, the use of methods that rely on the distance to surrounding geometry can be troublesome in indoor parts, as the user is too close to the geometry (e.g., walls and floors). Freitag et al. [27] solve this problem by using Viewpoint Quality Estimation (VQE) algorithms [29] to compute navigation speed according to the amount of relevant information on the virtual scene. This algorithm is based on the concept that scenes with higher viewpoint quality (high detailed rooms, for example) have more interesting aspects to see and objects to avoid, requiring low speeds and scenes with low viewpoint quality (empty corridors, for example), have less objects to collide and require higher speeds.

These algorithms also have another problem: the render bottleneck, as the algorithm needs to determine optimal speed by computing distances between the objects and the user in each frame. Taunay et al. [19] solve this problem by pre-processing the scene in a spatial structure such as kd-tree, that is rapidly accessed in run-time to determine user's velocity at a given frame. To overcome performance issues, the authors successfully use a combined GPU and CPU solution to determine optimal speed. This approach considers both visible and not visible objects to solve the navigation problem.

That work also presents an extensive user evaluation, which shows that its use is recommended for both experienced and inexperienced users. The problem of this approach is the inefficiency of navigating in dynamic scenes, where the kd-tree needs to be reconstructed at each frame. Hildebrandt et al. [30] overcome this issue by using parallel computing with the use of a service-oriented architecture on an image-based geographical navigation. However, this solution is still limited to image-based solutions, thus not suitable for general 3D scenes.

In the present paper, we aim to combine both approaches – the spatial partitioning heuristics [19] and parallel computing [30] – to improve efficiency of previous works by using an approach suited for both dynamic and static 3D virtual environments.

III. Remote Multiscale Network System : Automatic Speed Adjustment as a Service

In this work, we aim at improving the limitation obtained by the work proposed by our previous work [19], which was limited to static scenes. Previous work completely shifted the automatic speed adjustment calculation from the GPU to the CPU, so we are no longer tied to a mandatory local solution, i.e., when the nearest point was obtained from the render process, it necessarily had to be done on a machine where the entire 3D scene is being rendered. This limitation does not apply when dealing with an abstract point grid only in the CPU.

A natural instinct would be to separate the multiscale calculation into its own thread, and allowing us to increase our number of points budget without worrying about impacting other CPU processes (e.g., scene graphs). However, this would still limit any practical solution to a single programming language or specific framework, without any necessity. The multiscale speed adjustment is a separate abstract representation of any given scene and has no need to be even in the same machine.

In order to offer a completely generic and agnostic solution to the multiscale navigation problem, we decided to create a service dedicated exclusively for solving it, allowing not only local but also remote access. In a nutshell, the service would allow any consumer to register points to it, populating a remote *k-d* tree, and later on querying which optimal velocity should be used with a given point in space and camera frustum.

This solution was inspired by the microservice architecture [31], where systems are broken into several distributed services focused on offering a solution for a single problem. This mindset follows the popular Unix philosophy of “do one thing, and do it well”, and not only is effective for ensuring modularity between components, but also conveniently fits well into the current cloud oriented direction the tech industry tending toward.

Therefore, we developed an agnostic, isolated and scalable service that offers an API for obtaining the optimal multiscale speed solution for any given 3D scene, which uses the heuristic presented by Taunay et al. [19]. In this chapter, we will dive into the technical specifics of the RMNS service solution, present our performance results, and examine what other known problems this proposed solution helps us solve.

The RMNS is an open source initiative and all the source code, along with its documentation, tests and examples, is currently available at GitHub [32].

A. Architecture

Our server solution, which we named Remote Multiscale Navigation System (RMNS), is responsible for:

- Receiving and registering information relevant for automatically determining the navigation speed of a given scene
- Answering which is the optimal velocity for navigation with the given inputs and previously registered information

The server is separated in two layers. The top layer runs on Node.js’s [33] javascript V8 engine, which is responsible for data checking, high-level logic and all HTTP communication. This top layer binds seamlessly with the lower-level layer running on C++ process, which is responsible for all expensive geometric computation. All communication between both layers are asynchronous in order to avoid bottlenecks. See Figure 1.

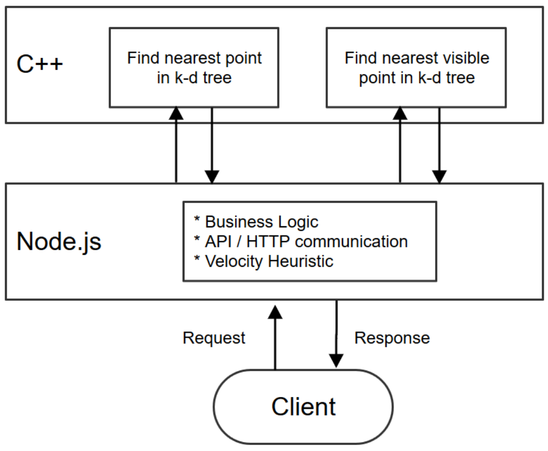
C++
Find nearest point in k-d tree
Find nearest visible point in k-d tree
Node.js
* Business Logic
* API / HTTP communication
* Velocity Heuristic
Request
Response
Client

Fig. 1: RMNS architecture.

Custom settings such as the KD-tree cell grid size and field of view (FOV) reduction - if any - are made by configuration and cannot be updated during execution.

A server can act following one of three different roles: stand-alone, master or slave. The former, as the name implies, is an independent approach where the entire system runs on a single Node.js process. The latter two are complementary when working with a distributed solution. They allow, for example, that while one slave only deals with finding the nearest point, another can deal with just finding the nearest visible point, therefore parallelizing both efforts. In this scenario the master’s role involves managing the communication between the slaves, as well as dealing with all business logic and the heuristic calculation.

The distributed approach leaves room for scaling scene complexities as well. Two different processes could be responsible for finding the nearest point in each half the scene, leaving the master to decide later on which one is closest. This improvement however could not be tackled during in time for this work and is postponed for future versions.

B. *Dealing with Dynamic Objects*

A downside from the CPU oriented nearest point solution is dynamic object support. While the GPU can seamlessly answer which is the nearest point in a given frame, without even having to be aware which objects are dynamic or not, rebuilding the *k-d* tree every frame is completely unfeasible for the CPU solution. This limitation did not pass unnoticed during the research and was given a lot of thought.

However, with a distributed system in place, we no longer need to be tied to the *k-d* tree exclusively. Imagine if while one process calculates the nearest point with the already known heuristics [19], a second process would calculate the nearest point taking into consideration only a relatively small subset of primitive objects (e.g. spheres and/or cubes). Figure 2 attempts to illustrate such a solution:

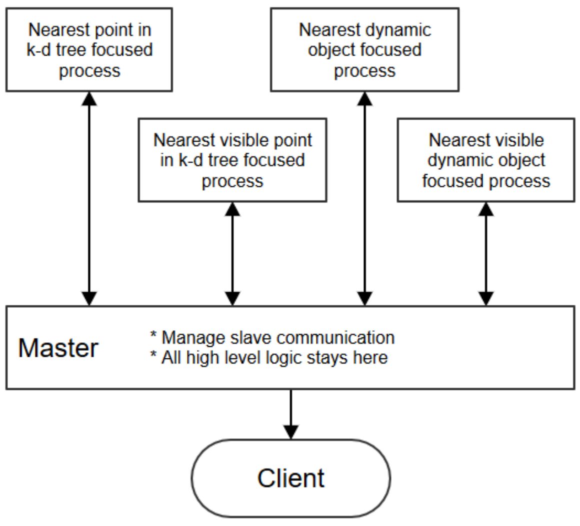
Nearest point in k-d tree focused process
Nearest dynamic object focused process
Nearest visible point in k-d tree focused process
Nearest visible dynamic object focused process
Master
* Manage slave communication
* All high level logic stays here
Client

Fig. 2: Distributed heuristic approach.

The advantages of using primitive objects are that they are cheaper to re-register on a frequent basis (data transfer wise) and do not demand rebuilding any spatial structure. Instead of transmitting several thousand points (or more) with a sphere we would need only a center and radius, for example. These primitive objects are stored in a list that on every request would be iterated linearly storing the nearest point found on each object, and returning only the closest one of all. Currently only spheres are supported as dynamic objects.

Since we are dealing with a list, and will be iterating it linearly, a reasonable suspicion could be raised over scalability issues. A benchmark was conducted with a mid-range server in order to measure this approaches performance, which proved that the bottleneck was not the linear loop as would be expected, which even with 1M spheres managed to maintain less than 40ms necessary taking into account both nearest global and visible points, but the limit of the body of the HTTP request package itself. In other words, the system can - within an acceptable time budget - register/update spheres in the scale of millions and on top of them calculate the optimal velocity heuristic, the issue is being able to update all sphere positions when an HTTP request can usually only transfer sphere data in the scale of tens of thousands. A workaround can be sending multiple sphere registration requests at a given frame, but in this case no test is needed to acknowledge that this approach would definitely not scale. Therefore, this dynamic-object solution is currently limited to a scale range of tens of thousands of objects.

It's worth noting that running RMNS in a distributed topology is not mandatory for dealing with scenes with dynamic objects. The stand alone mode also offers this feature, and if the scene's complexity and the machine's processing power allow it, navigation works seamlessly.

C. *Consuming the API*

All calls to the RMNS are by design asynchronous. This may lead to unfamiliar scenarios in computer graphics applications, such as a later call returning before a previous one. In a scenario where the current optimal speed is raising or decreasing in a constant ratio, responses that return from the server out of order may lead to a shaking and unstable navigation. A solution for this problem is to return in every answer a timestamp, making the client responsible for verifying and eventually ignoring if any answers are already deprecated. The figure 3 exemplifies this situation.

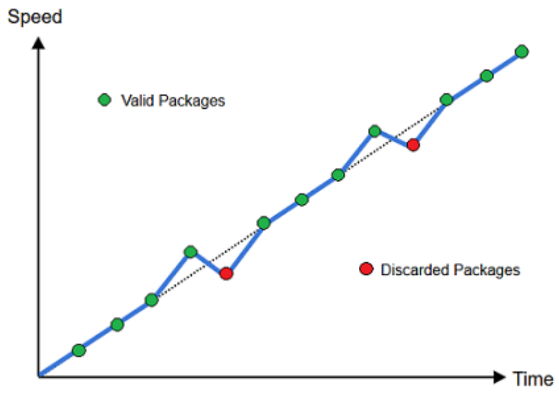

Fig. 3: Navigation shaking for out of out responses.

Currently there is support for point and basic geometry registration. The latter can, and should, be updated frequently during navigation, while the former must be registered previous to interaction, as rebuilding the point spatial structure in real-time is not supported. Any calls made to the point registration APIs during interaction will return an error response.

D. Solar System Experiment

As a demo test for the RMNS, we created a 3D scene representing the solar system, where each planet is represented by a sphere geometry, with their positions being updated on a frequent basis as dynamic objects, and the asteroid belt between Venus and Jupiter being represented by a 2M static size point cloud. The theme was chosen given the multiscale nature of the scene, that is also well known by the wide public. Figure 4 displays a screen-shot of the demo, which runs in the Unity3D engine (with C# code), proving the agnostic nature of the service. The demo can also be found at the project's web page [32], and is a good reference as an example on how to consume the RMNS.

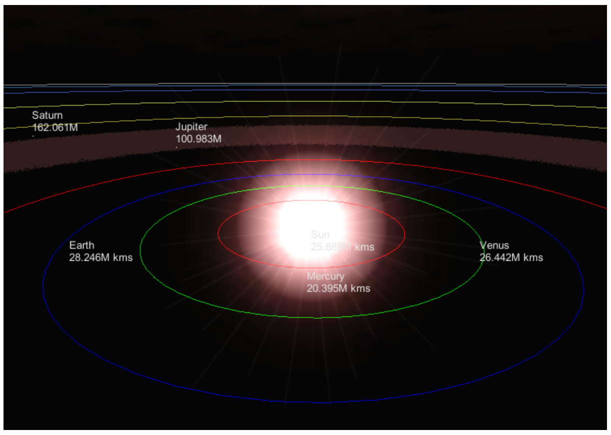

Fig. 4: Solar system demo.

E. Performance

The time taken for the RMNS to answer the optimal navigation speed is formed by the round-trip time plus the processing time on the server side. Taking into consideration the scenario where each process is run in parallel, as well as the fact that the optimal velocity can only be achieved when each of its answers are made available, we can conclude that the service is as fast as its slowest slave answer, plus the round-trip time. Figure 5 illustrates this behavior. In this example, processes A, B and C could be the nearest global point, the nearest visible point and the nearest visible and global sphere processes respectively, or any other combination of distributed processes as the system's users see fit. In fact, there could be more or less than three processes, since the service's generic architecture allows any number of distributed setups. Each distribution configuration should be fine-tuned depending on each scene's nature, targeting the minimization of the lengthiest process.

We have managed to obtain a 140ms answer time performance accessing from Rio de Janeiro, Brazil, a server running the RMNS in a data center in Texas, USA. From the 140ms total time, 20ms were from the round-trip and 120ms from

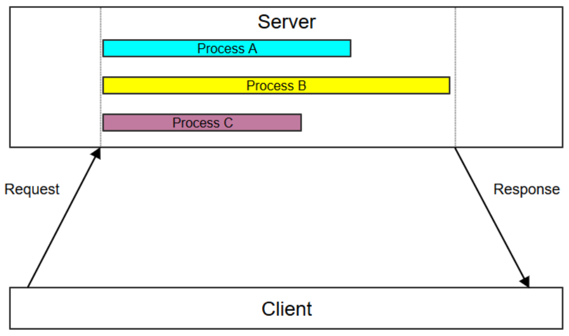

Fig. 5: Round-trip path.

the processing time bottleneck. This result was achieved with a single server. When distributing the service with a master server and two slaves performing the nearest point heuristic, a 90ms roundtrip time was achieved. In the distributed scenario, the bottleneck was the visible nearest point process, with an average 70ms processing time, that when added with the 20ms round-trip we reach the 90ms mark. Since RMNS still can't break a scene's k-d tree into separate processes this bottleneck currently cannot be any more parallelized. Now, despite that 140ms - or even 90ms - can be considered high-processing times in traditional synchronous computer graphic applications, the system's asynchronous nature does not affect in any way the main process, and therefore the roundtrip time is only relevant regarding how much it affects the user's navigation experience. During the tests, the delay did not appear to present any significant different - positive or negative - to the navigation experience. All performance tests were run working with an interval of approximately 10 requests per second, with a number of points in the scale of millions (4M to be exact) and spheres in the scale of thousands (5k to be exact), on top of mid-range virtual machines.

IV. Conclusion

Navigating in a multiscale environment is still a difficult and unnatural task to perform. Commonly, researchers propose techniques that analyze the scene and optimally determine speed in order to facilitate navigation. However, analyzing such scenes is a CPU and GPU-intensive procedure and may not be practical for rendering effective navigation in most complex scenes in real-time. Other solutions use spatial structures to pre-process the scene and determine speed in real-time, but the cost of pre-processing the scene make this solution not possible to deal with dynamic scenes, where objects are in motion. We overcome this problem by proposing RMNS (Remote Multiscale Navigation System), that uses a service distributed approach to tackle the automatic speed adjustment problem. RMNS is an initiative to make the solution available to the scientific community and an universal solution for navigating in dynamic multiscale virtual environments. This system is an open-source solution, which enables for parties interested in understanding or contributing towards the service.

It also succeeds in isolating the problem from the main navigation system, providing an high level and language agnostic architecture interface, while also isolating the automatic speed velocity computation by design. We were also able to suggest a simple and complementary alternative by working with basic geometries that, while not being as generic as the previous GPU approach, can be proven useful depending on the scene being dealt with.

For future works, we intend to explore the idea of breaking the scene into separate sub-scenes, in order to allow multiple processes to answer the nearest-point question in parallel. This feature will prove necessary the moment we start dealing with larger and more complex scenes where the grid strategy will not be able to reduce the total number of points enough to gain efficiency. We also intend to include support to more basic geometries - i.e., cubes and capsules - with the objective of offering more fine tuning and versatility when dealing with dynamic objects. Finally, on a different front, we plan to study the possibility of working with spatial structures that could allow reconstruction in real time, e.g., a more efficient variation of the *k*-d tree, or a BVH (bounding volume hierarchy). In the case of the BVH, it not only may prove helpful for dealing with dynamic objects, but it also has potential to be used as an alternative to the grid structure built during the pre-processing phase, with the goal of maintaining part of the multiscale nature of a given scene.

Acknowledgment

The authors would thank CAPES and CNPq for the continuous support.

References

[1] B. B. Bederson, L. Stead, and J. D. Hollan, "Pad++: Advances in multiscale interfaces," in *Conference companion on Human factors in computing systems*. ACM, 1994, pp. 315–316.

[2] K. Perlin and D. Fox, "Pad: an alternative approach to the computer interface," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, 1993, pp. 57–64.

[3] G. W. Furnas and B. B. Bederson, "Space-scale diagrams: Understanding multiscale interfaces," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 1995, pp. 234–241.

[4] J. J. LaViola Jr, D. A. Feliz, D. F. Keefe, and R. C. Zeleznik, "Hands-free multi-scale navigation in virtual environments," in *Proceedings of the 2001 symposium on Interactive 3D graphics*. ACM, 2001, pp. 9–15.

[5] F. Bacim, D. Bowman, and M. Pinho, "Wayfinding techniques for multiscale virtual environments," in *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*. IEEE, 2009, pp. 67–74.

[6] R. Kopper, T. Ni, D. A. Bowman, and M. Pinho, "Design and evaluation of navigation techniques for multiscale virtual environments," in *IEEE Virtual Reality Conference (VR 2006)*. IEEE, 2006, pp. 175–182.

[7] N. J. McFarlane, X. Ma, G. J. Clapworthy, N. Bessis, and D. Testi, "A survey and classification of visualisation in multiscale biomedical applications," in *2012 16th International Conference on Information Visualisation*. IEEE, 2012, pp. 561–566.

[8] D. Song and M. L. Norman, "Looking in, looking out: Exploring multiscale data with virtual reality," *IEEE Computational Science & Engineering*, vol. 1, no. 3, pp. 53–64, 1994.

[9] X. Zhang, "Multiscale traveling: crossing the boundary between space and scale," *Virtual Reality*, vol. 13, no. 2, pp. 101–115, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10055-009-0114-5

[10] F. Carvalho, D. R. Trindade, P. F. Dam, A. Raposo, and I. H. dos Santos, "Dynamic adjustment of stereo parameters for virtual reality tools," in *Virtual Reality (SVR), 2011 XIII Symposium on*. IEEE, 2011, pp. 66–72.

[11] D. R. Trindade and A. B. Raposo, "Improving 3d navigation techniques in multiscale environments: a cubemap-based approach," *Multimedia Tools and Applications*, vol. 73, no. 2, pp. 939–959, 2014.

[12] A. Bönsch, S. Freitag, and T. W. Kuhlen, "Automatic generation of world in miniatures for realistic architectural immersive virtual environments," in *IEEE Virtual Reality Conference*, 2016, pp. 155–156.

[13] J. S. Pierce and R. Pausch, "Navigation with place representations and visible landmarks," in *Virtual Reality, 2004. Proceedings. IEEE*. IEEE, 2004, pp. 173–288.

[14] R. Stoakley, M. J. Conway, and R. Pausch, "Virtual reality on a wim: interactive worlds in miniature," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 1995, pp. 265–272.

[15] P. Abtahi, M. Gonzalez-Franco, E. Ofek, and A. Steed, "I'm a giant: Walking in large virtual environments at high speed gains," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: ACM, 2019, pp. 522:1–522:13. [Online]. Available: http://doi.acm.org/10.1145/3290605.3300752

[16] X. Zhang and G. W. Furnas, "mcves: using cross-scale collaboration to support user interaction with multiscale structures," *Presence: Teleoperators and Virtual Environments*, vol. 14, no. 1, pp. 31–46, 2005.

[17] R. Kopper, T. Ni, D. A. Bowman, and M. Pinho, "Design and evaluation of navigation techniques for multiscale virtual environments," in *VR '06: Proceedings of the IEEE Virtual Reality Conference (VR 2006)*. Washington, DC, USA: IEEE Computer Society, 2006, p. 24. [Online]. Available: http://dx.doi.org/10.1109%2FVR.2006.47

[18] F. Argelaguet and M. Morgan, "Adaptive navigation for virtual environments," in *IEEE Symposium on 3D User Interfaces*, 2016, pp. 91–94.

[19] H. Taunay, V. Rodrigues, R. Braga, P. Elias, L. Reis, and A. Raposo, "A spatial partitioning heuristic for automatic adjustment of the 3d navigation speed in multiscale virtual environments," in *3D User Interfaces (3DUI), 2015 IEEE Symposium on*. IEEE, 2015, pp. 51–58.

[20] J. McCrae, I. Mordatch, M. Glueck, and A. Khan, "Multiscale 3d navigation," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, ser. I3D '09. New York, NY, USA: ACM, 2009, pp. 7–14. [Online]. Available: http://doi.acm.org/10.1145/1507149.1507151

[21] T. Foley and J. Sugerman, "KD-tree acceleration structures for a GPU raytracer," in *Graphics Hardware*, M. Meißner and B.-O. Schneider, Eds. Los Angeles, California: Eurographics Association, 2005, pp. 15–22. [Online]. Available: http://www.eg.org/EG/DL/WS/EGGH/EGGH05/015-022.pdf

[22] D. R. Horn, J. Sugerman, M. Houston, and P. Hanrahan, "Interactive k-d tree GPU raytracing," in *SI3D*, B. Gooch and P.-P. J. Sloan, Eds. ACM, 2007, pp. 167–174. [Online]. Available: http://doi.acm.org/10.1145/1230100.1230129

[23] G. W. Fitzmaurice, J. Matejka, I. Mordatch, A. Khan, and G. Kurtenbach, "Safe 3D navigation," in *Proceedings of the 2008 Symposium on Interactive 3D Graphics, SI3D 2008, February 15-17, 2008, Redwood City, CA, USA*, E. Haines and M. McGuire, Eds. ACM, 2008, pp. 7–15. [Online]. Available: http://doi.acm.org/10.1145/1342250.1342252

[24] J. D. Mackinlay, S. K. Card, and G. G. Robertson, "Rapid controlled movement through a virtual 3d workspace," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '90. New York, NY, USA: ACM, 1990, pp. 171–176. [Online]. Available: http://doi.acm.org/10.1145/97879.97898

[25] F. Argelaguet and C. Andujar, "Automatic speed graph generation for predefined camera paths," in *International Symposium on Smart Graphics*. Springer, 2010, pp. 115–126.

[26] C. Moerman, D. Marchal, and L. Grisoni, "Drag'n go: Simple and fast navigation in virtual environment," in *3D User Interfaces (3DUI), 2012 IEEE Symposium on*. IEEE, 2012, pp. 15–18.

[27] S. Freitag, B. Weyers, and T. W. Kuhlen, "Automatic Speed Adjustment for Travel through Immersive Virtual Environments based on Viewpoint Quality," in *2016 IEEE Symposium on 3D User Interfaces (3DUI)*, March 2016, pp. 67–70.

[28] Q. Duan, J. Gong, W. Li, S. Shen, and R. Li, "Improved cubemap model for 3d navigation in geo-virtual reality," *International Journal of Digital Earth*, vol. 8, no. 11, pp. 877–900, 2015.

[29] S. Freitag, B. Weyers, A. Bönsch, and T. W. Kuhlen, "Comparison and evaluation of viewpoint quality estimation algorithms for immersive virtual environments," in *Proceedings of the 25th International Confer-*

ence on Artificial Reality and Telexistence and the 20th Eurographics Symposium on Virtual Environments, ICAT-EGVE, vol. 2015, 2015.

[30] D. Hildebrandt and R. Timm, "An assisting, constrained 3d navigation technique for multiscale virtual 3d city models," *GeoInformatica*, vol. 18, no. 3, pp. 537–567, 2014.

[31] M. Fowler and J. Lewis, "Microservices," *ThoughtWorks. http://martinfowler. com/articles/microservices. html [last accessed on February 17, 2015]*, 2014.

[32] H. Taunay. Github : Rmns. [Online]. Available: https://github.com/htaunay/rmns

[33] R. Dahl, "Node. js: Evented i/o for v8 javascript," *URL: https://www. nodejs. org*, 2012.