Delft University of Technology
CSE3000 – Research Project – Final Report (2023)
R. Bidarra and M. Skrodzki (Editors)

Short Paper

Rendering Non-Euclidean Space in Virtual Reality Using Portals

R. Slotboom[1]

[1]Delft University of Technology

Abstract

Simulating non-Euclidean geometry in virtual reality is of interest to a wide variety of fields of research. However it is still quite a challenge. Various methods are already known, but they vary greatly in performance and applicability. This paper compares some methods of rendering a non-Euclidean space. We focus on the order-5 square tiling that can be found in the hyperbolic plane, but the methods used are also relevant to other non-Euclidean spaces. We render this space using portals with two different implementations: one using render textures, an one using stencil polygons. Through an experiment where we measured and compared the frame rate of each method, we have found that, even with a small number of portals, the stencil polygon approach is more than two times as efficient. However, this method is limited in the number of portals, whereas render textures can be used for any number of portals.

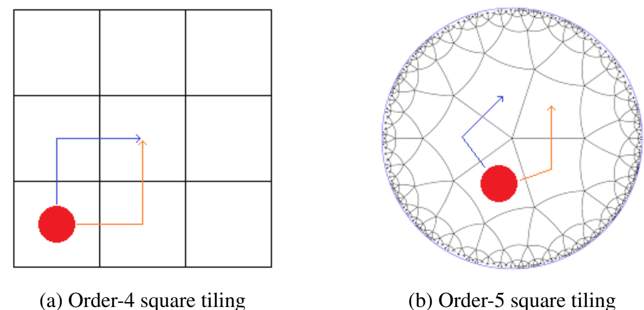
1. Introduction

Over two millennia ago, Euclid's five postulates laid the foundation for geometry. These postulates reflect the world around us, and it was not until the 19th century that mathematicians managed to describe alternative geometries that do not follow Euclid's postulates. However, visualizing these non-Euclidean geometries was still a challenge, as they are inherently impossible to fully recreate in the real world. Nowadays however, we can use computer graphics to simulate various types and parts of non-Euclidean geometries, in order to get a better grasp of how they work.

Hyperbolic geometry is one example of non-Euclidean geometry. Because the available space grows exponentially when moving away from the origin, it is an interesting tool for fields like data visualization. It may also enable us to investigate how a person learns new modes of navigation. However to do this effectively, the user must be able to immerse themselves in the non-Euclidean world. For this it is important to render non-Euclidean space appropriately.

To investigate how to render non-Euclidean geometry, we will use an experimental virtual reality game called `Holonomy` [YBS*22]. The game, made in the Unity game engine as part of the Software Project course in 2022, is based on an order-5 square tiling, where five square tiles are placed around each vertex rather than the four that we are used to in Euclidean space, as shown in Figure 1b. Such a tiling exists in the hyperbolic plane, and was chosen for `Holonomy` as a simplified model of hyperbolic space for reasons that we will discuss in Section 3. The player navigates this non-Euclidean game world by physically moving around in a 3x3 play area.

Rendering this tiling using a Euclidean render engine is challenging because different tiles will need to occupy the same area of Euclidean space, as seen in Figure 1. One way of rendering this is to divide the grid into "subgrids", and have "portals" linking the subgrids to the main grid. That is, if two tiles would occupy the same Euclidean space, then two subgrids are created, along with two portals that each show one of the two subgrids. An example layout is shown in Figure 2.



(a) Order-4 square tiling (b) Order-5 square tiling

Figure 1: Example of how different tiles in an order-5 square tiling can occupy the same space in a Euclidean order-4 square tiling. In the Euclidean tiling, following the blue arrow takes the player (represented by the red circle) to the same tile as following the orange arrow. However, in the order-5 square tiling, these two arrows lead to two different tiles.

One of the biggest challenges in `Holonomy`'s rendering is that there are multiple portals, some of which are placed behind or in the same position as other portals. As seen in Figure 2, if the player is standing in one of the corner tiles, then there are two tiles in the center which are seen through portals, and beyond that (in the opposite corner) there are a total of four tiles which are seen through

© 2023 The Author(s)

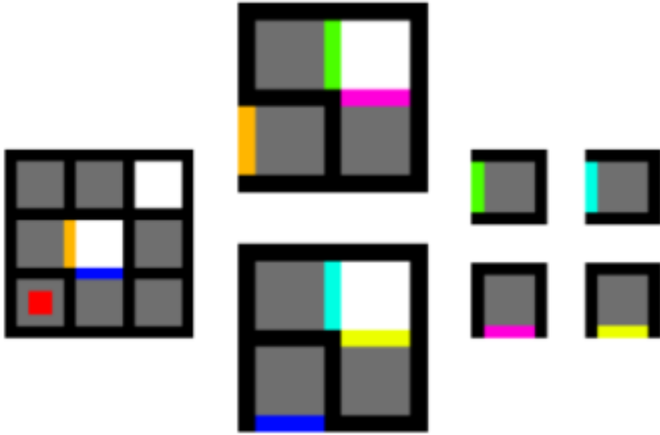


Figure 2: An example layout for rendering an order-5 square tiling in a 3x3 grid. In this example, the player is represented by the red square in the bottom-left tile of the main grid. A total of six subgrids are created, and when the player looks through a portal (represented by colored edges), they will see the corresponding subgrid (with the same colored edge in it). [YBS*22]

two levels of portals. Thus for each approach, we will also consider how the portals can be made to interact with each other in the desired way.

This paper aims to answer the following question: how can we render a non-Euclidean space based on an order-5 square tiling in a performance-friendly manner? To answer this question, the following will be investigated:

- What methods of rendering hyperbolic space are there?
- How do these methods compare to each other in terms of performance?
- What technical limitations exist for these methods?

2. Background

This section outlines existing solutions to related problems of non-Euclidean rendering. An approach using render textures is explained in Section 2.1. Section 2.2 then explains a method using stencil polygons.

2.1. Render Textures

One possible method of rendering non-euclidean space uses render textures. This method creates portals by first rendering the portal view onto a texture, and then rendering a quad with that texture as the portal itself [AL97].

This approach using render textures can also be extended to work recursively, by scaling the created texture to fit the portal inside the portal view [KK18]. Another option is to continuously move the camera that is used to render the texture, rendering the deepest level of recursion first and inserting the texture before rendering the next deepest level. The first recursion method is useful to emulate infinite recursion, where a portal is contained within its own portal

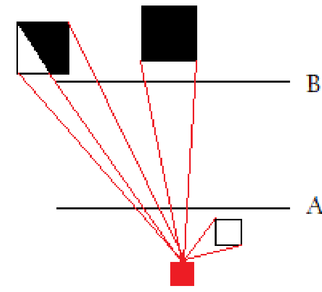


Figure 3: An example setup for stencil polygon portals. The portals themselves are represented by the lines marked "A" and "B". The camera is represented by the red square. Some rays are shown coming from the camera to an object. Before reaching portal A, the stencil value of a ray will be 0. Portal A sets it to 1. Finally, when the ray reaches portal B, the value is set to 2, which in this example is the requirement for the geometry (black boxes) to be rendered. The shaded parts of boxes are the parts that are actually rendered.

view. The second method, which is the method previously implemented in `Holonomy` [YBS*22], is useful for finite recursion with multiple portals that see each other, but not themselves.

2.2. Stencil Polygons

Another approach for creating portals is to use stencil polygons [NHH*20] rather than render textures. Stencil polygons are transparent objects that write a value to the stencil buffer. Any objects behind these polygons then read from the stencil buffer to determine if they should be rendered. This allows two objects to be placed in the same position in a way that the first is seen through one stencil polygon and the second through another stencil polygon.

In Unity, stencil polygons work as follows. Each object – both the stencil polygon and the actual geometry that should be rendered – is rendered using a shader with a stencil test [Pet13]. The stencil test consists of a comparison function, such as "Equal" or "Less", and a reference value, which is an integer. The value in the stencil buffer is compared to the reference value using the given function. If this test passes, the object will be rendered. Additionally, the shader dictates what operation is performed on the stencil buffer after the test. Examples of operations are "Keep" to leave the value unchanged, or "Replace" to set the buffer to the same reference value that was used for comparison. This operation can differ based on whether the stencil test passes. For example, it is possible to perform the "Keep" operation if the test passes and the "Replace" operation if it fails. An example of how stencil polygons work is shown in Figure 3

3. Related Work

`HyperRogue` [KCC17] is a game set on the hyperbolic plane. The player is shown a top-down view of the game world, rendered using

© 2023 The Author(s)

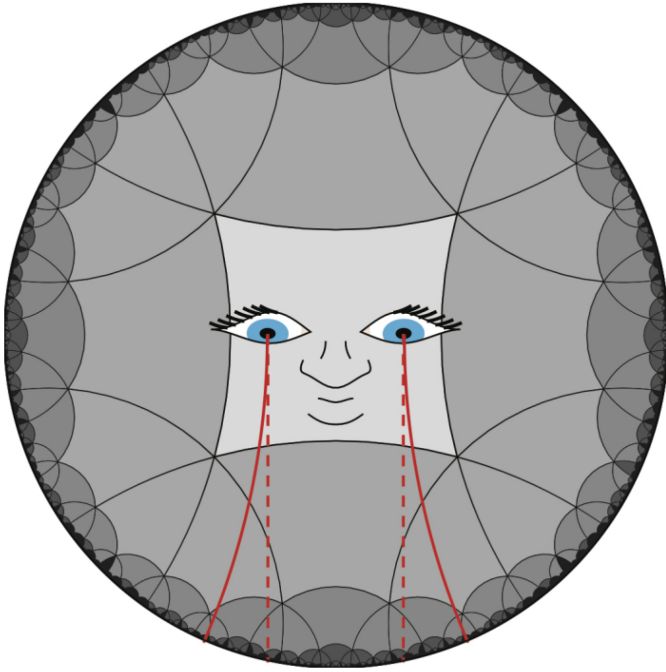


Figure 4: Diverging light rays from two eyes looking straight ahead in $\mathbb{H}^2$. [HHMS17b]

the Poincaré disk model. HyperRogue gives us an idea of what it is like to navigate tilings of the hyperbolic plane, but because it is played in two dimensions it does not allow the player to fully immerse themselves in a non-Euclidean world.

A more recent game utilizing hyperbolic geometry is Hyperbolica [Cod22], which does use three dimensions, unlike the previous example. Specifically, it is set in $\mathbb{H}^2 \times \mathbb{E}$. That is, the product of the hyperbolic plane and a Euclidean vertical axis. The game's implementation uses gyrovector spaces as a hyperbolic analogy to the vector spaces that are widely used for Euclidean rendering [Ung05, Ung04]. Hyperbolica even supports virtual reality, although this is not "true" hyperbolic virtual reality as we do not experience the phenomenon of geodesic deviation.

Geodesic deviation is one of the properties that makes hyperbolic rendering challenging, in particular in virtual reality. [HHMS17a, Skr20]. A geodesic is a straight path through two points; in Euclidean space, this is simply a line. However in hyperbolic space, two geodesics that start in the same direction will diverge. This means that light rays coming into our eyes also diverge as they travel along geodesics (see Figure 4), and we will see something different than what we are used to from $\mathbb{E}^3$.

Geodesic deviation and other properties of hyperbolic space cause unpleasant experiences for a player in a virtual reality game [Wee21], which is why "true" hyperbolic virtual reality is not feasible. This issue is the main reason why Holonomy is based on an order-5 square tiling but otherwise uses Euclidean rendering.

4. Method

As mentioned in Section 1, having multiple portals on the screen simultaneously poses some additional challenges. This section explains some ways of overcoming those challenges when working with stencil polygons. Section 4.1 goes into the challenges of placing portals behind other portals. Section 4.2 shows how to solve the issues that arise when multiple stencil polygons are placed at the same position.

4.1. Multiple Stencil Polygons in Sequence

When multiple portals are placed in sequence, those on the first level of depth can be implemented in the same way as when they are the only portals in the scene (except there may be some constraints on the utilized reference values, as we will see in the following paragraphs). Any portals that are placed deeper do require a different implementation: they should only be rendered if the stencil value from the first level is equal to some reference value. If this test passes, they should change the stencil value to some other value. This is where the problem arises: we need one reference value for reading and another for writing, but comparison and modification both use the same reference value in the Unity engine, which is used for Holonomy.

A simple solution would be to use two stencil polygons for each portal beyond the first level, instead of just one. The first polygon compares the stencil buffer to the desired value and writes the value 0 if the test fails[†]. The second polygon then writes the new stencil value, but only if the stencil value is currently greater. If the first level of depth is set to use stencil values greater than the second level of depth, this comparison will always succeed unless the buffer was set to 0 by the first polygon.

Auxiliary stencil polygons are not always desirable, so we will consider another possible solution. We can only apply this solution in a case where the first level of depth only has two stencil polygons. On this level, polygon A will be set to use the smallest out of all the utilized stencil values, and B will use the largest. Then at the second layer of depth, portals that should be seen through A can test if the buffer is smaller than their reference value, and those that should be seen through B can test if the buffer is larger than their reference value. Again, this will not work if the first level has more than three possible values, since there is no way to distinguish between all three using only "less than" and "greater than" comparisons. Additionally, if more than two levels of depth are used, this same restriction applies to each level except the final level.

A third solution, again without any auxiliary stencil polygons, works using the "increment" operation of the stencil shader. If stencil polygons C and D should be seen through stencil polygon A, then both C and D first compare the stencil buffer to the desired reference value (that of A). If the test passes, C keeps the current value and D increments the value. This method can distinguish any number of stencil polygons on the first level of depth, but is limited

† At first glance this still seems to require two reference values in the same stencil polygon, but there exists a dedicated operation for writing 0 to the stencil buffer which can be used here.

© 2023 The Author(s)

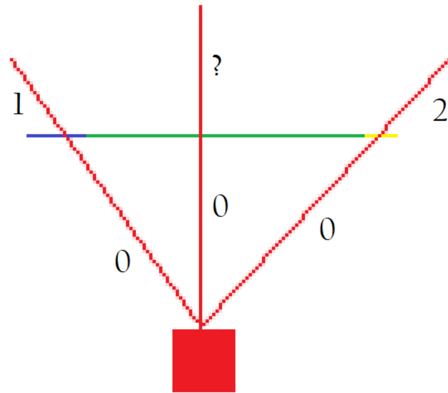


Figure 5: Conflicting stencil polygon portals. The blue portal uses reference value 1, the yellow portal uses reference value 2, and the green line represents an overlap between the two portals. Each segment of a camera ray is marked with the corresponding stencil value. The final value of the middle ray is not known, as it depends on the render order of the portals; whichever portal renders last dictates what the value will be.

to two portals[‡] at the second level per portal on the first level. It also would not work for more than two levels of depth; at the third level, incrementing the value of C is the same as keeping the value of D, so these can no longer be distinguished in this way.

4.2. Multiple Stencil Polygons at the Same Position

If multiple portals are placed at the same position, there are some additional challenges in implementing them using stencil polygons. These polygons, if implemented incorrectly, may overwrite each other's stencil values, causing the wrong geometry to be rendered behind it. Such a conflict is illustrated in Figure 5.

Firstly, this means that the first of the potential solutions in 4.1 will not work. To see why, take two stencil polygons C and D that are placed at the same position in the second level of depth. Here C should be seen only through A, and D only through B. If the test of C's auxiliary polygon fails, the buffer will be set to 0. Then the test of D's auxiliary polygon will automatically fail, even when seen through B. Thus in the use case of `Holonomy`, one of the other methods must be implemented.

Secondly, the operation of one polygon should not interfere with the comparison of another. This becomes relevant for the second proposed solution: there should be no overlap between the range seen through A (which is compared using "less than") and the range

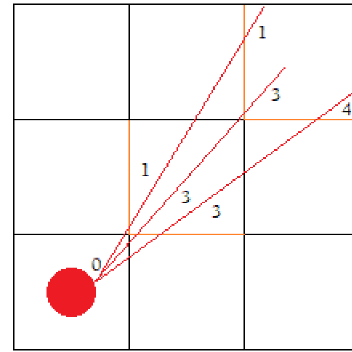


Figure 6: An example of how stencil polygons are implemented in our game. Some example camera rays are shown, with each segment being labeled with the stencil buffer value as the ray passes through stencil polygons (marked in orange).

seen through B (compared using "greater than"). This does not invalidate the method, but must be kept in mind when implementing it.

5. Implementation

Section 4.1 proposed three methods for portals behind other portals using stencil polygons. The first method cannot be applied to `Holonomy` because auxiliary stencil polygons would not interact with each other correctly (see Section 4.2). The other two methods are similar to each other, both in implementation and in performance, since they use the same number of stencil polygons. Since the third method is more general than the second, we implemented this one in `Holonomy`. The first level of portals uses reference values 1 and 3, and the second level either keeps or increments these values, giving us a range of 1 to 4 as our reference values. This is illustrated in Figure 6.

In order to ensure the correct render order, different render queues were used in Unity's universal render pipeline. This allows us to always render the first level of portals before the second, and lets us render geometry after the corresponding portals are rendered. Within each render queue, objects are rendered farthest to nearest, using render queues that are usually used to ensure that transparent objects are rendered in the correct order.

6. Evaluation

This section aims to compare the render texture approach and the stencil polygon approach, to see which is more effective for rendering our order-5 square tiling. Section 6.1 explains an experimental setup to analyze the performance of these methods, and Section 6.2 shows the results of this experiment. Section 6.3 then discusses what can be concluded from these results.

6.1. Methodology

In order to compare the performance of the stencil polygon approach to that of the render texture approach, we have conducted the following experiment, using the Unity editor to run

[‡] This method can be extended by utilizing the "decrement" operation in addition to the "increment" and "keep" operations, but this still limits the number of available portals.

© 2023 The Author(s)

the project. A camera was placed in the center of one of the tiles in Holonomy's 3x3 grid. The camera then made 3 full rotations around the Y axis over the span of 12 seconds. Each frame, we measure the time since the frame before it, and take the reciprocal of this time to find the number of frames per second (FPS).

The portal layout in Holonomy depends on the player's position within the grid. This leads to three distinct layouts: one where the player is in the center tile, one where they are in a corner tile, and one where they are in an edge tile. We conducted the above experiment separately for each of these cases. For the last two cases, there is not always a portal in the camera's field of view, because it will be pointed straight at the nearest wall for some duration. Because of this, we also repeated the experiment with a smaller range of camera angles, only letting it rotate 90 (for a corner tile) or 180 (for an edge tile) degrees before reversing the direction and repeating.

It should be noted that this experiment was not done in Holonomy itself due to conflicts with existing assets that would have taken too long to resolve for this project. Instead, we made a separate Unity project in which we manually replicated each of the previously mentioned portal layouts. This is still representative of the actual performance since we are only interested in the comparison of the two methods.

6.2. Results

Table 1 shows the results of the experiment in Section 6.1 for render textures, and Table 2 shows the results for stencil polygons. Note that in all cases (either approach, any tile, and any range of angles), one of the first few frames has a value very close to 1 frame per second. This is likely caused by the loading time of the Unity editor and thus not representative of the actual results. For this reason, we ignore the initial second of frames when listing the minimum FPS. Finally, Table 3 shows the speedup of the stencil polygon method, obtained by dividing its average FPS by that of the render texture approach.

6.3. Discussion of Results

As expected and mentioned in Section 6.1, limiting the camera's movement to always show at least one portal significantly lowers the FPS of the render texture method, seen in the average and standard deviation in Table 1. However, for stencil polygons (Table 2) this is not the case. This indicates that stencil polygons add little to no overhead with additional portals, whereas render textures do.

We also see that for render textures, the corner tile case has a lower FPS than the other cases[§]. There are two possible explanations for this. Firstly, this case is the only one that has portals behind other portals, and thus the only case requiring recursion. Secondly, while the total number of portals is lower than in the center tile case, more portals are shown simultaneously. That is, when standing in a corner tile, all six portals can be seen, but from the center tile we can only see four of the eight portals at a time. Again, this decrease in FPS does not happen for stencil polygons because of the low overhead.

Table 3 shows the speedup that stencil polygons provide. If we disregard the cases where the camera is pointed at the wall for most of the time, we see that stencil polygons perform about 2.3 times faster, or even more for the case described above. This makes them significantly more efficient, and as we have seen this difference will be even bigger when there are more portals in the scene.

While stencil polygons clearly have a better performance than render textures, there are some limitations to the number of stencil polygons and the maximum depth, as explained in Section 4. The alternative method using auxiliary polygons does not have these constraints, but stops working when multiple polygons are placed at the same position, as is the case in Holonomy. This approach is also limited in the depth of recursion [LD03]. This is not a problem for Holonomy itself, since it only requires two layers of depth, but may pose a problem for other use cases or for future versions of the game. This is a clear downside compared to the render texture approach, as render textures can be rendered recursively to any level of depth.

The render texture approach is limited when it comes to lighting. Objects that are rendered behind the portals are not lit by light sources in front of the portals, and vice versa. Therefore it may happen that the player sees a light source in an adjacent room, but does not see this light in the room they are currently occupying. With stencil polygons however, light sources behave mostly as expected. We did not investigate if light sources can also be filtered out using stencil polygons, which may still be investigated in future work.

7. Responsible Research

The methods used are explained in Section 6.1 to ensure that the experiment can be reproduced. In our discussion of the results (Section 6.3), we state our conclusions and clearly show how the data supports those conclusions. We also consider the validity of our test cases – in particular the corner tile and edge tile cases, where the camera is pointed at the wall for much of the duration.

It is important to consider how players experience the Holonomy game, even in a paper that does not perform experiments with human participants. In Section 3, we have seen why we chose to use our order-5 square tiling model rather than true hyperbolic geometry. Additionally, improving the frame rate of the portals is the main focus of our research. Both of these points serve to prevent motion sickness for players of Holonomy.

8. Conclusion

Our experiments have shown that stencil polygons perform significantly better than render textures for rendering our portals. Furthermore, our tests used only a small number of portals, and adding more portals impacts the performance of render textures much more than it does stencil polygons, making the latter more scalable. However, this scalability does eventually reach an upper limit to the number of portals, whereas render textures work with any number. In short, render textures are less efficient, but more generally applicable than stencil polygons.

[§] Only when limiting the movement to 90 degrees, because when making full rotations the camera will point at the wall for roughly 75% of the duration.

© 2023 The Author(s)

	Total number of portals	Average	Minimum after one second	Maximum	Standard deviation
Center tile	8	143.843830	13.072681	209.894423	34.19565984
Corner tile	6	258.604970	17.555625	409.718523	92.5478929
Corner tile (90 degrees)	6	92.67712857	9.16414729	148.9669144	30.36676437
Edge tile	4	216.967100	18.209497	396.652255	87.75802708
Edge tile (180 degrees)	4	146.4999928	13.08432981	207.1851821	36.87121221

Table 1: Average, minimum, maximum, and standard deviation of the measured FPS for render texture portals.

	Total number of portals	Average	Minimum after one second	Maximum	Standard deviation
Center tile	8	333.679134	21.542346	447.487358	74.0470264
Corner tile	6	332.965919	7.593937	465.311060	76.97215996
Corner tile (90 degrees)	6	367.6417286	23.61704489	487.3531849	70.6103923
Edge tile	4	313.425702	11.469639	434.442610	67.58017243
Edge tile (180 degrees)	4	342.3625739	24.48621793	460.4263548	72.48499617

Table 2: Average, minimum, maximum, and standard deviation of the measured FPS for stencil polygon portals.

There are still some questions left unanswered by this paper, since this project was relatively limited in time and therefore misses some interesting points. These points could be investigated in future work.

Firstly, only two methods were investigated in this paper. Stencil polygons and render textures are not the only methods of rendering hyperbolic space, and other methods can be compared to these two using a setup similar to the one used here.

Secondly, the performance of the two approaches was compared only in terms of rendering time. Another metric that may be used in the context of `Holonomy` is the time required to generate the grid when the player moves into a new tile. The new implementation requires fewer objects to be placed, so there may be a speedup in this part of the game as well.

Lastly, what we did not consider is a combination of our two methods – using render textures for some subset of our portals, and stencil polygons for the rest. By combining them, we could make the stencil polygons as generally applicable as render textures. This would come at the cost of some of the efficiency we gain from stencil polygons, but would still be much more efficient than using exclusively render textures. However the optimal strategy for combining the two methods depends heavily on the use case, as we would want to minimize the number of render texture portals while eliminating the constraints posed by the stencil polygons.

References

[AL97] ALIAGA D., LASTRA A.: Architectural walkthroughs using portal textures. In *Proceedings. Visualization '97 (Cat. No. 97CB36155)* (1997), pp. 355–362. doi:10.1109/VISUAL.1997.663903. 2

[Cod22] CODEPARADE: Hyperbolica, 2022. URL: https://store.steampowered.com/app/1256230/Hyperbolica/. 3

[HHMS17a] HART V., HAWKSLEY A., MATSUMOTO E., SEGERMAN H.: Non-euclidean virtual reality i: Explorations of hÂ³. In *Proceedings of Bridges 2017: Mathematics, Art, Music, Architecture, Education, Culture* (Phoenix, Arizona, 2017), Swart D., Séquin C. H., Fenyvesi K., (Eds.), Tessellations Publishing, pp. 33–40. URL: http://archive.bridgesmathart.org/2017/bridges2017-33.html. 3

[HHMS17b] HART V., HAWKSLEY A., MATSUMOTO E. A., SEGERMAN H.: Non-euclidean virtual reality ii: explorations of $\mathbb{H}^2 \times \mathbb{E}$, 2017. arXiv:1702.04862. 3

[KCC17] KOPCZYNSKI E., CELINSKA D., CTRNACT M.: Hyperrogue: Playing with hyperbolic geometry. In *Proceedings of Bridges 2017: Mathematics, Art, Music, Architecture, Education, Culture* (Phoenix, Arizona, 2017), Swart D., Séquin C. H., Fenyvesi K., (Eds.), Tessellations Publishing, pp. 9–16. URL: http://archive.bridgesmathart.org/2017/bridges2017-9.html. 2

[KK18] KIRCHER D., KOHLI T.: Portal problems, 2018. URL: https://cs50.harvard.edu/games/2018/weeks/11/. 2

[LD03] LOWE N., DATTA A.: A fragment culling technique for rendering arbitrary portals. In *Computational Science — ICCS 2003* (Berlin, Heidelberg, 2003), Sloot P. M. A., Abramson D., Bogdanov A. V., Dongarra J. J., Zomaya A. Y., Gorbachev Y. E., (Eds.), Springer Berlin Heidelberg, pp. 915–924. 5

[NHH*20] NEERDAL J. A. I. H., HANSEN T. B., HANSEN N. B., BONITA K. L. F., KRAUS M.: Navigating procedurally generated overt self-overlapping environments in vr. In *Interactivity, Game Creation, Design, Learning, and Innovation* (Cham, 2020), Brooks A., Brooks E. I., (Eds.), Springer International Publishing, pp. 244–260. 2

[Pet13] PETERSSON A.: Fast complex transformative portals, 2013. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A830875&dswid=-6058. 2

[Skr20] SKRODZKI M.: Illustrations of non-euclidean geometry in virtual reality, 2020. arXiv:2008.01363. 3

[Ung04] UNGAR A.: On the unification of hyperbolic and euclidean geometry. *Complex Variables, Theory and Application: An International Journal 49*, 3 (2004), 197–213. URL: https://doi.org/10.1080/02781070310001657976, arXiv:https://doi.org/10.1080/02781070310001657976, doi:10.1080/02781070310001657976. 3

[Ung05] UNGAR A.: Gyrovector spaces and their differential geometry. *Nonlinear Functional Analysis and Applications 10* (01 2005). 3

[Wee21] WEEKS J.: Body coherence in curved-space virtual reality games. *Computers Graphics 97* (2021), 28–41. URL: https://www.sciencedirect.com/science/article/pii/S0097849321000443, doi:https://doi.org/10.1016/j.cag.2021.04.002. 3

[YBS*22] YARAR B., BAKKER B., SNELLENBERG R., SLOTBOOM R., LI W.: *"Holonomy": a non-Euclidean labyrinth game in virtual reality.* Tech. rep., TU Delft, 2022. URL: http://resolver.tudelft.nl/uuid:60d473f0-f327-411e-a402-95d44e27f088. 1, 2

© 2023 The Author(s)

	Average FPS for render textures	Average FPS for stencil polygons	Speedup factor
Center tile	143.843830	333.679134	2.32
Corner tile	258.604970	332.965919	1.29
Corner tile (90 degrees)	92.67712857	367.6417286	3.97
Edge tile	216.967100	313.425702	1.44
Edge tile (180 degrees)	146.4999928	342.3625739	2.34

Table 3: Speedup factor for five test cases, obtained by dividing the average FPS of the stencil polygon portals by that of the render texture portals.

© 2023 The Author(s)