Transparency-Based 4D Visualization in Virtual Reality

Jacob Malin
malin146@umn.edu
University of Minnesota - Twin Cities
Minneapolis, Minnesota, USA

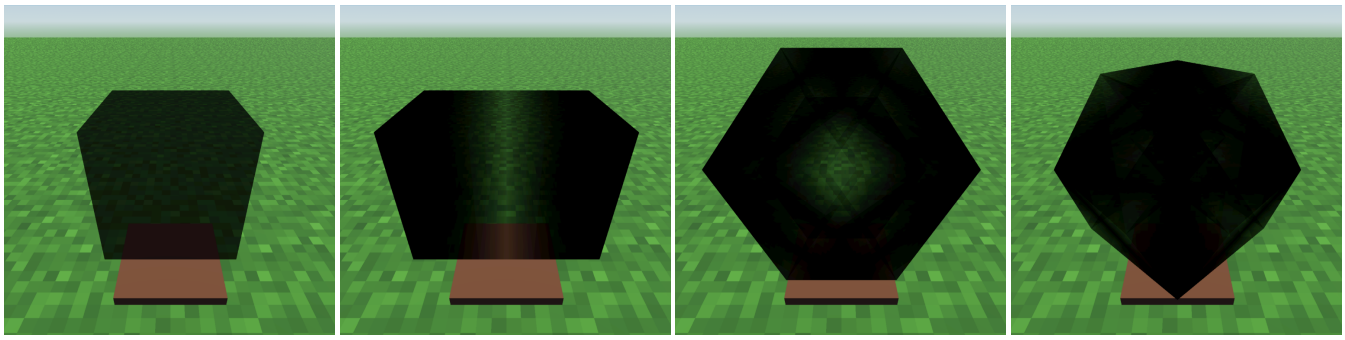

Figure 1: Some possible projections of a hypercube into a 3D space. From left to right, a cube, a cuboid, a hexagonal prism, and a rhombic dodecahedron.

ABSTRACT

As virtual reality is a three-dimensional display, it has a natural advantage over two-dimensional displays when visualizing four-dimensional objects. The purpose of this project is to give users a more intuitive understanding of 4D objects and 4D rotation by using VR. This 4D visualization is done as a first-person orthographic projection. To better display objects that are not in the viewing space, transparency is utilised to enable users to view objects up to 1 meter away from the player camera. Points intersecting the viewing space are opaque, and decrease in transparency as distance from the camera increases, until past 1 meter away where objects are not visible. To enable users to see more than just the xyz-space, users can rotate the viewing space to capture new perspectives of 4D objects. The six degrees of freedom required to rotate the viewing space are split so that three degrees are controlled by the rotation of the head and three degrees are controlled by a 3D mouse that is emulated using a VR controller. To further help the user to understand 4D concepts, a one-dimension-down model has also been created. Instead of a 3D observer being placed within a 4D body, the model has a 2D observer placed within a 3D body, in which the user can manipulate the 3D body to observe how the 2D observer's view changes as they move and as the viewing plane of the observer is rotated using the 3D mouse.

CCS CONCEPTS

• **Human-centered computing** → *Interaction techniques*; **Scientific visualization**; • **Computing methodologies** → **Virtual reality**.

KEYWORDS

4D, Virtual Reality, Visualization, Hypercubes

ACM Reference Format:
Jacob Malin. 2024. Transparency-Based 4D Visualization in Virtual Reality. In . ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As virtual reality (VR) technology advances, it allows for new insight into what it means to view a screen, and how information can be visualized. With flat two-dimensional (2D) rendering, the user's camera remains fixed within physical space, while the scene appears to move in and out of the camera's view. However, with three-dimensional (3D) VR rendering, the user's camera moves within the physical space, and the scene appears fixed. This work makes use of the user's natural intuition when viewing 3D VR scenes to visualize four-dimensional (4D) objects in a fashion more comprehensible to the user than viewing from a 2D screen.

This work discusses a method of visualizing 4D objects and their rotation by taking advantage of VR and the unique 3D input devices available to VR. This work implements a transparency-based system to indicate distance from the camera in the w-axis, and an input system modeled after the normal 2D computer mouse, however taking advantage of the 3D tracking of VR controllers. This work was implemented in the Godot game engine for the Meta Quest 2.

4D as discussed in this paper is the theoretical fourth dimension of space, in which in addition to the x, y, and z-axes there is also the w-axis. In 3D, there are 6 degrees of freedom (DOF); 3 DOF of translation, and 3 DOF of rotation. In 4D, there are 10 DOF, 4 translational DOF with the additional degree being the w-translation, and 6 rotational DOF: xy, yz, zx, xw, yw, and zw. An important note about rotation in 4D is that rotation occurs around planes rather than axes.

Other important terms that will come up in this paper are space, cell, and hypercube. Space in the next term in the sequence discussing elementary geometric elements: point, line, plane, then space. As one vector can define a line, two can a plane, it takes three vectors to define a space. The space that all 3D rendering uses is the xyz-space. A cell is the next term in the sequence discussing topological elements: vertex, edge, face, and cell. In which a cube has 8 vertices, 12 edges, 6 faces, and 1 cell. Finally, a hypercube, sometimes referred to as a tesseract, is the next term in the sequence of fundamental geometric shapes: point, line segment, square, cube, and hypercube. In which a hypercube consists of 8 cells, each of which is a cube, with one cube in each positive and negative axis direction.

2 RELATED WORK

Many works have explored the concept of 4D visualization, due to the inherent difficulty of understanding 4D concepts without visuals. For example, Chu et al. made a 4D renderer named GL4D, which uses a tetrahedron-based rendering pipeline rather than a triangle-based pipeline [3].

In another work, David Banks in his dissertation proposed and implemented a system to use transparency to indicate distance in the w-direction [2]. Banks also describes that his ideal input scheme uses four 3D controllers to control the 10 DOF required for 4D input. Notably, Banks also cites the mindset of Hoffman, who believes that it is important to decouple 4D manipulations from the more familiar 3D manipulations [7], to allow the user to naturally interact with the system, which is closer to the mindset that was employed for this work.

There have also been many works implementing 4D visualization in XR. For example, Collins et al. implemented a wireframe orthographic projection of a hypercube using an HTC Vive [5]. Or Prabhat et al. who used a Cave system to teach 4D rotation [10]. However, since viewing 4D objects in 3D must reduce one dimension, each method has the possibility to vary greatly in implementation. For example, some works use stereographic vs orthographic projection and some works emphasize different aspects of the visualization to draw the user's attention such as coloring the cells of the hypercube differently [8] or providing only a wireframe [5].

Some of the more interesting cases of 4D visualization come from games. For example, 4D Golf, which has the user playing golf in 4D, gives the user a novel sense of immersion and understanding of 4D scenes [4]. Or, 4D Miner, an as-of-yet unreleased game that takes the familiar voxel-based sandbox game and instead uses hypercubes as the basic unit [9].

There have also been games released with VR compatibility, for example, 4D Toys, which has a user manipulating brightly colored 4D objects that are physically simulated [11]. Or, Magic Cube 4D VR, which allows a user to scramble and solve a 4D Rubik's Cube [6].

3 ACCOMPLISHMENTS

3.1 Tiny's Table

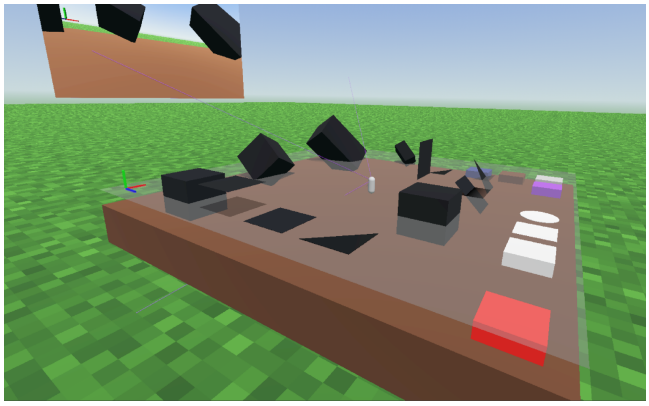

Figure 2: A one-dimension-reduced model consisting of a 2D character, named Tiny, placed within a 3D character body, depicted as a gray capsule. Also pictured is Tiny's viewing plane which is a transparent gray. Tiny's 3D body is facing towards the left, which is shown by a purple view frustum.

3.1.1 Flatland. It is easiest to explain the fourth dimension using the Flatland metaphor [1], in which the user first observes a lower-dimension character in the same situation that they will soon be in. As such, I created a lower-dimension model in which a 2D character named Tiny is placed within a 3D body.

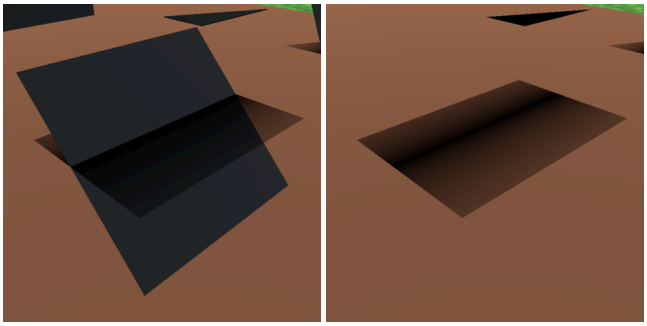

Figure 3: The same square with both 3D and 2D representations on the left, and only the 2D projection on the right.

Tiny, as a Flatlander, can only view one plane, in this case, the horizontal plane, which in Godot is the xz-plane. To enable Tiny to see objects that are not within Tiny's viewing plane, with inspiration from Banks' work [2], transparency is used to indicate distance from the viewing plane. A different pattern of transparency is used from Banks' work, however. Objects intersecting the viewing plane are at full opacity, and as the y-distance from the plane increases, the transparency is linearly interpolated until a certain distance away from the viewing plane, at which objects are rendered fully invisible. For Tiny, the visible distance from the viewing plane is 0.1 meters. This can be seen in figure 3, which shows that the 3D shape intersects the most opaque region, and the edges further from the horizontal plane taper off. Notably, it is impossible to tell the difference between positive and negative y-direction using this method.

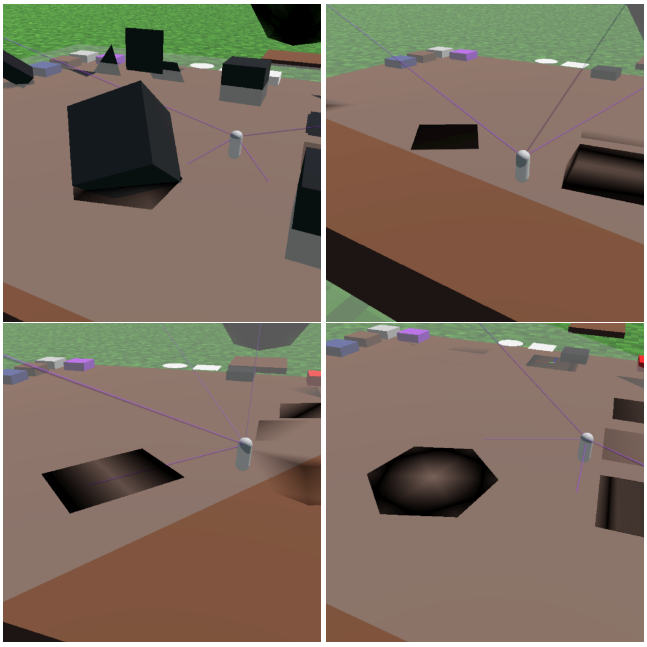

Figure 4: The top left cube is the reference 3D object. The other three images show different rotations of the viewing plane; a square in the top right, a rectangle in the bottom left, and a hexagon in the bottom right.

3.1.2 The Viewing Plane. Other than being able to view 3D objects using transparency, Tiny has also been enabled to rotate their viewing plane. While in Tiny movement mode, which is toggled by one of the table buttons, the user can hold the grip button and move the controller to rotate Tiny's body's view. Taking inspiration from the traditional 2D computer mouse mapping of rotation, forwards and backward will pitch the view, and left and right will yaw the view. In addition, and unlike a 2D mouse, up and down motion will roll the view. Taking the metaphor of the 3D mouse further, all input is considered locally, that is, moving the controller forwards will produce the same rotation as rotating the controller to point up then moving the controller upwards. To avoid clutching, and also taking inspiration from the mouse metaphor, mouse acceleration was added so that moving the controller quickly will result in large rotations, while slow movement will result in precise rotation. An example of viewing plane rotations can be seen in figure 4.

3.1.3 Rendering and Projection. The 2D rendering/projection method is as follows, each vertex is orthographically projected to the viewing plane and the distance to the viewing plane is recorded. Then each vertex is rotated around Tiny's camera so that the final vertices are in line with the xz-plane. Finally, based on the recorded distance from the viewing plane, transparency is applied in the method mentioned previously. This is all done within a shader to save on processing.

All shapes within this simulator are made up of triangles, so they are all face-only meshes. Squares were created by pairing two triangles to form a square. Cubes were created first offsetting the six squares from the center of rotation, then rotating each square

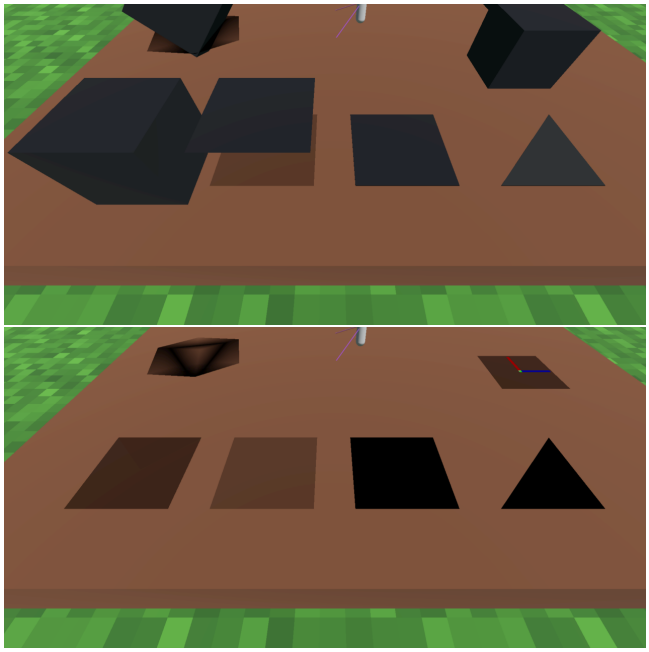

Figure 5: The construction of a cube. On top is the 3D version, and on bottom, is the 2D visualization. From right to left in both images, a triangle, a square, a square displaced in the +y-direction, and a cube. In the bottom image, the cube appears to be a slightly transparent square because all but two faces are perpendicular to the xz-plane.

so that each was placed perpendicular to a positive or negative axis. This construction can be seen in figure 5, along with how this appears with the 2D projection.

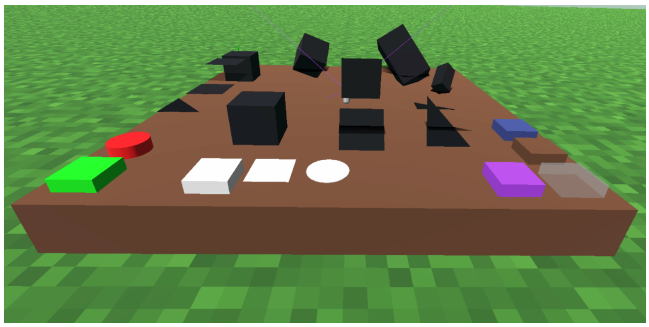

Figure 6: The buttons on Tiny's Table. From back to front, left to right: the view reset button, the enable movement toggle, the 3D shape visibility toggle, the 2D shape visibility toggle, the projection rotation toggle, the frustum visibility toggle, the screen visibility toggle, the table visibility toggle, and Tiny's view plane visibility toggle.

3.1.4 Table Buttons. Several buttons were built into the table to facilitate the presentation and examination of Tiny's Table, see figure 6. The enable movement button swaps movement and rotation

from the player character to Tiny's body. While this is active, the user is unable to move. The view reset button resets Tiny's viewing plane to the xz-plane. The 3D and 2D shape visibility toggles do as the name describes. The projection rotation toggle changes from the projected 2D shapes being aligned with the xz-plane and Tiny's viewing plane; this is useful for explaining how the projection functions. The four remaining buttons are visibility toggles for Tiny's body's frustum, Tiny's body's camera's screen, the brown table mesh, and Tiny's viewing plane. Notably disabling the table mesh will also disable the collision allowing for closer inspection of the objects on the table.

3.2 4D Visualization

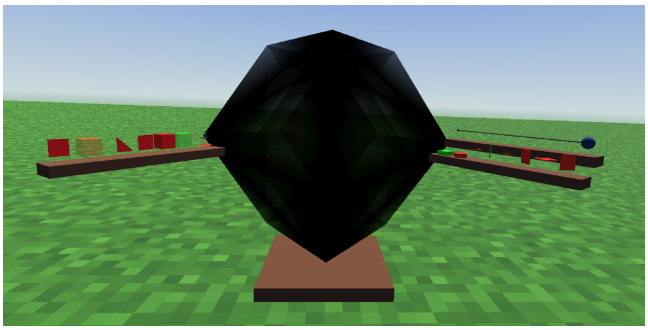

Figure 7: The 4D visualization pedestal. Pictured is a hypercube and the control panels for control over the display.

3.2.1 4D Transformation. To support 4D shapes, it was necessary to create a base class to handle the 4D transform matrix. The 4D transformation matrix, *TRS*, consists of the translation, rotation, and scaling matrices. The translation and scaling matrices scale as expected from their 3D counterparts:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 & t_y \\ 0 & 0 & 1 & 0 & t_z \\ 0 & 0 & 0 & 1 & t_w \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad S = \begin{bmatrix} s_x & 0 & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 & 0 \\ 0 & 0 & s_z & 0 & 0 \\ 0 & 0 & 0 & s_w & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The rotation matrix is more complicated consisting of six fundamental rotations, one for each rotational DOF:

$$R = \begin{bmatrix} R_{zw}R_{xw}R_{yw}R_{zx}R_{yz}R_{xy} & 0 \\ 0 & 1 \end{bmatrix}$$

The six rotation matrices, as specified by Zhang [12] are:

$$R_{xy} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_{zw} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$R_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_{xw} = \begin{bmatrix} \cos\theta & 0 & 0 & \sin\theta \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta & 0 & 0 & \cos\theta \end{bmatrix}$$

$$R_{zx} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_{yw} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta \\ 0 & 0 & 1 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

A given rotation matrix is named R_{ab} where a and b are the axes that are not in the plane of rotation. For example, R_{xy} rotates around the zw-plane. This means that R_{yz}, R_{zx}, R_{xy} correspond respectively to the 3D rotation matrices R_x, R_y, R_z.

Since Godot uses a yxz rotation order, the rotation order (zw, xw, yw, zx, yz, xy) was decided by first maintaining the original rotation order and then layering the new rotations onto the front. As I could not find a standard 4D rotation order, the new rotations were added in an order that looked satisfying when rotating a triangle in the game engine.

The rotation matrices are precomputed as one large rotation matrix for purposes of optimization. As well, matrix multiplication is computed within a shader rather than on the CPU whenever possible.

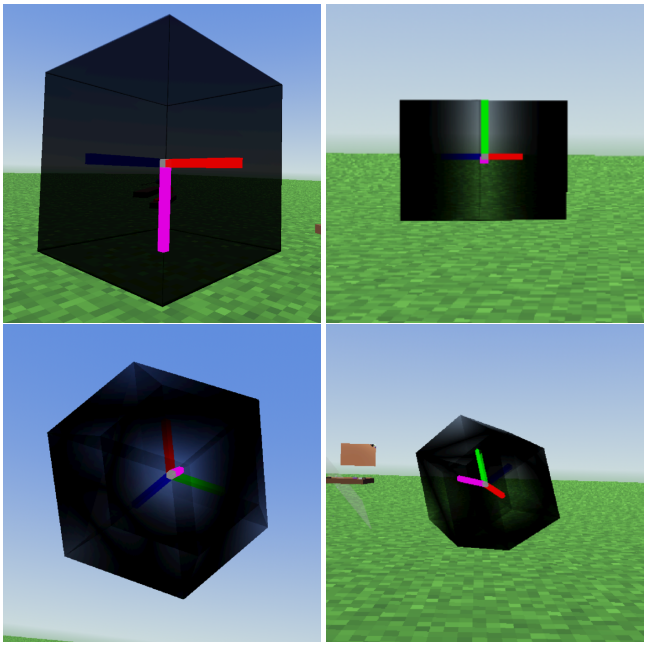

Figure 8: Four different rotations of the viewing space. The origin of the axes is in the center of the hypercube. The red, green, blue, and purple axes are, respectively, the x, y, z, and w axes. The images show, in reading order, a cube, a cuboid, a hexagonal prism, and a very sheared rhombic dodecahedron.

3.2.2 The Viewing Space. Just as Tiny was able to rotate their viewing plane, the user has been enabled to rotate their viewing space. Since there are six rotational DOF, the input space was split in half, with the more recognizable 3D rotations mapped to the rotation of the HMD, and the other rotations being mapped to the right controller similar to Tiny. When the grip button is held, forwards and backward will rotate around the xy-plane, left and

right will rotate around the yz-plane, and up and down will rotate around the xz-plane. An example of viewing space rotations can be seen in figure 8.

3.2.3 Rendering and Projection. The 3D rendering/projection method is very similar to the 2D method, with everything happening one dimension up. Each vertex is first orthographically projected to the viewing space and the distance to the viewing space is recorded. Then each vertex is rotated around the main camera to be in line with the xyz-space. Finally, based on the recorded distance from the viewing space, transparency is applied so that objects are visible up to 1 meter away from the user. This is all done within a shader to save on processing.

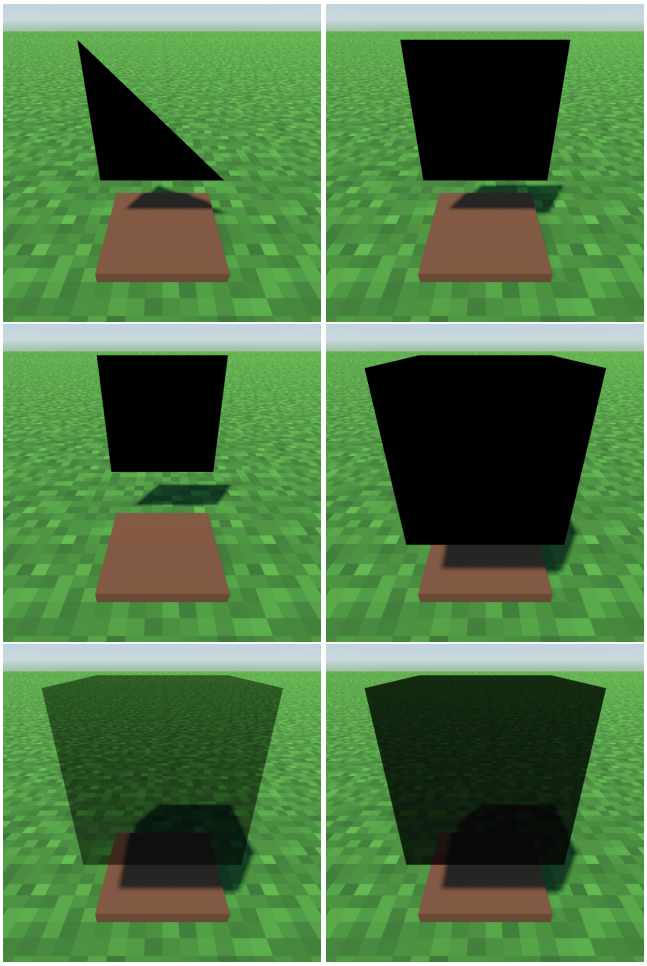

Figure 9: The construction of a hypercube. In reading order, a triangle, a square, a square displaced in the +z-direction, a cube, a cube displaced in the +w-direction, and a hypercube. The hypercube appears to be a slightly transparent cube because all but two cells are perpendicular to the xyz-space.

Similar to Tiny's table, all 4D shapes are constructed from triangles, which means that these shapes are made of only faces. In a similar fashion, starting with triangles, two are paired together to make a square. Then that square is offset duplicated and rotated to 6 positive/negative axes. Then that cube is offset, duplicated, and rotated to each of the 8 positive/negative axes. However, since these are face-only meshes, this causes each face in the hypercube to be doubled. So, as a final step, half of the faces are disabled within the hypercube to both save on rendering and to create the correct transparency effect. This construction can be seen in figure 9.

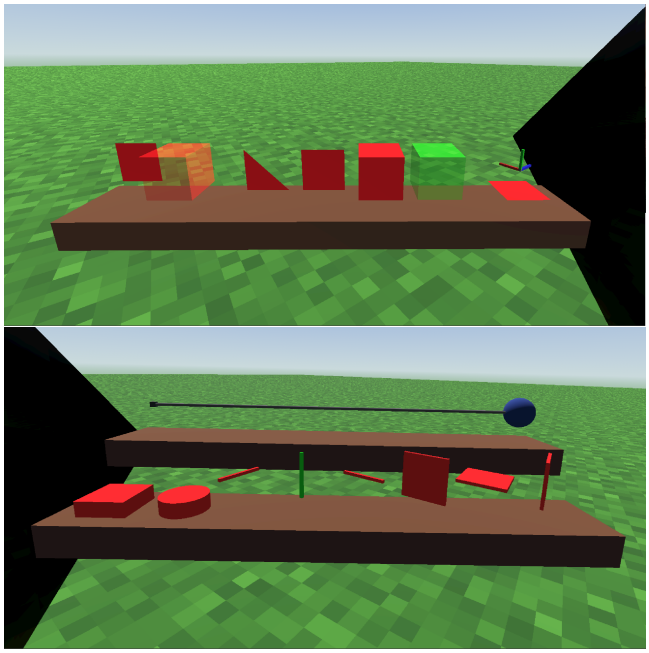

Figure 10: The top image shows the left control panel and the bottom image shows the two right control panels.

3.2.4 Control Panel. The control panels can be seen in figure 10. The left control panel controls for the display of shapes. From left to right, the first two buttons are the buttons that push the square in the +z-direction and that push the cube in the +w-direction. These are used to present the construction of a hypercube. The next four buttons are the buttons that show a triangle, a square, a cube, and a hypercube, respectively. The final button shows a four-color axes.

The top right control panel has a slider that can adjust how far away from the camera the user can see in the w-axis. By default this is 1 meter, but can be lowered to as low as 0.01 meters. This effect can be seen in figure 11. The bottom right control panel has the controls for rotation of the shape displayed. From left to right, the first two buttons control the pause/play of the rotation and the rotation reset. The next six buttons control what rotation is playing, from left to right, rotation around the xw, yw, zw, yz, zx, and xy planes.

4 ASSESSMENT

4.1 Optimization

One of my major hurdles that I overcame in this project was optimization. I had to rewrite major portions of this code to allow

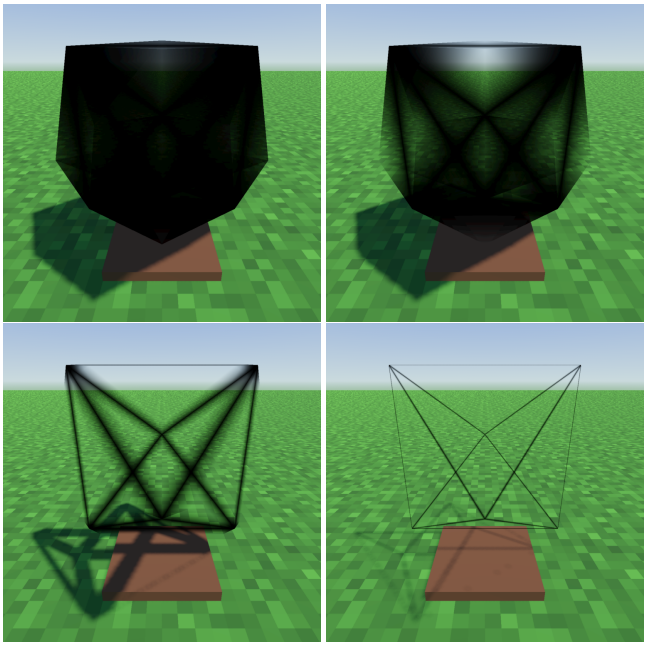

Figure 11: Various w-distances from the camera. In reading order, 1 meter, 0.5 meters, 0.1 meters, and 0.01 meters. We can see that while the shape appears to be a rhombic dodecahedron, the portion passing through the viewing space is an octahedron.

my final demo consisting of six hypercubes to run smoothly. These rewrites consisted of moving any and all logic of the projection into shaders to improve rendering, as well as convert said shaders into instance shaders to created as much shared code as possible. Even after that, matrix multiplication was still a large issue, so much work was done to reduce or eliminate matrix multiplication, including preventing the recalculation of matrices that do not change over time, and precomputing the rotation matrices to reduce multiplication at runtime. While I am proud of the precomputed 4D rotation matrix, it is highly dependent on the rotation order, so I have not included it here.

4.2 User Input

Adding mouse acceleration to the rotation controls significantly improved the usability of the controls. Clutching was a major problem with the control scheme, and while there is still clutching necessary, it is much reduced. Mouse acceleration also allowed for a more intuitive understanding of how the rotation controls worked where after using the rotation controls on Tiny's table for a while I was able to reach past imagining it as three separate rotations and instead as how moving my hand corresponded to the position that I wanted. It also significantly improved my understanding of the 4D rotations, however I was not able to achieve an intuitive grasp of the rotation, though this could be due to the inherent difference of being able to view the 3D rotations from a third person perspective.

Adding the w-distance slider was one of the most useful UI elements that I added. The w-distance slider allowed a finer understanding of what portions of the shape pass through the viewing space. While this was easy to do in Tiny's table, the w-distance slider also allowed for this to be done in the 4D visualization. Also, at the minimum w-distance of 0.01 meters, it allows a view into how slicing projections relate to this transparency-based projection method. For example, I was unsure how my project related to a project such as 4D Toys [11] which shows the user a slice of the 4D world. However, once reducing the slider to 0.01 meters, I was able to observe similar behaviors between how shapes rotated in this project and how their hypercubes roll around, which left me with a much better understanding of how 4D Toys works.

As it took me significant time to implement I would like to mention here my UI design and the preparations I made for presenting this project. Many of the buttons implemented in this project attempt to use metaphors that take advantage of the computer generated aspect and hint at their usage, for example how the reset button on Tiny's table is only visible when the table is enabled; how the rotational axes are oriented in the correct direction; or how the buttons for toggling 2D/3D shapes are a square and a cube, respectively. Notably, there were a few controls that were duplicated onto the controllers for ease of use, such as toggling Tiny's movement, resetting rotation or either the viewing plane or space, and toggling rotation of the hypercube.

4.3 Parallels between the 3D and 4D Visualizations

There are many interesting parallels that arose between Tiny's table and the hypercube visualization that give interesting insight into the fourth dimension. For example, observing both figures 5 and 9, we can observe that the square being lifted in the y-direction and its resulting projection becoming more transparent which neatly parallels the cube being lifted in the w-direction and becoming more transparent. Since it is possible to observe that on Tiny's table between the 2D and 3D versions, it allows the mind to imagine what must be happening between the 3D and 4D versions in a way that I was not able to imagine previously. As well, looking at the bottom right image of figure 9, it is difficult to imagine how 6 of the 8 cells in the hypercube could be perpendicular to the xyz-space; However, it is easier to imagine when viewing the differences in the two images in figure 5 and how 4 of the faces in the cube are perpendicular to the xz-plane, and thus are not visible. As well, it becomes possible to extrapolate where those perpendicular cells must be, in that there should be one cell laying in the same space as each face of the visible cube.

There is also a very neat symmetry between figures 4 and 8. We can observe that it is possible at certain rotations of the viewing plane for a cube to appear as a square, a rectangle, or a hexagon. And similarly, we can observe that at certain rotations of the viewing space that a hypercube can be viewed as a cube, a cuboid, or as a hexagonal prism. To add to that, while not pictured, it is also possible for a hypercube to be viewed as a square, a rectangle, or a hexagon at certain rotations of the viewing space. This means that while a viewing plane can from a cube capture a square, a rectangle, and a hexagon; a viewing space can from a hypercube capture a

square, a rectangle, a hexagon, a cube, a cuboid, a hexagonal prism, and a rhombic dodecahedron. This shows the unimaginably greater span of 4D as opposed to 3D, as well as interesting information about the nature of primitive shapes and how they relate to each other.

Whilst not necessarily a parallel between 3D and 4D, there is a very interesting parallel between figures 1 and 8. Whilst both capturing the same sequence, cube, cuboid, hexagonal prism, rhombic dodecahedron; both are created using different means. Figure 1 is created by rotating the hypercube with a constant viewing space (in this case the xyz-space), while figure 8 is created by rotating the viewing space with a constant rotation of the hypercube (notably the euler angles: 0,0,0,0,0,0). This means that it is very likely that the method of rotating the viewing space covers the same possible rotations as rotating the hypercube directly. This raises interesting questions about what exactly makes the two method different from each other. As well, it raises interesting questions about what the positional differences between the two indicates about the hypercube. Notably, rotating the hypercube does not move the position of the hypercube, while rotating the viewing space can have drastic effects on the position of the hypercube, as viewed in figure 8 where the position of the hypercube is decidedly not on top of its pedestal in any of the images.

5 LIMITATIONS AND FUTURE WORK

5.1 Optimization

One of the major limitations was optimization. Only after optimizing the code greatly was it able to smoothly run six hypercubes simultaneously, and that is with a NVIDIA 4070 laptop graphics card and only on full power. Notably, each hypercube is only 48 triangles. This would most likely require a restructure from the ground up and to use a game engine that actually has 5x5 matrix multiplication, or in fact, any native matrix multiplication.

5.2 Improved modeling and rendering

One area of future work would be to implement the other graphics primitives such as toruses, cylinders, and spheres. This was not implemented due to time restrictions as well as optimization concerns. As well, support for a model format such as obj (with extensions to support the fourth coordinate) would do wonders to support general shape modeling, however most likely a model editor would also have to be built as few tools support the editing of 4D models.

In the theme for better modeling, 4D lighting, or even 3D lighting would be beneficial for understanding depth within the models. It would also be important to support normals and color within the models. However, the reason color was not supported in this version was because initial testing made it very hard to distinguish transparency when models were multi-colored.

One area for future work would be to explore how adding vertices and edges could help with understanding of the shapes. One important reason that this was not attempted in this version is due to the triangle rendering limit. Notably, this project was a face-only view of a 4D model, but it would also be very interesting to observe and edge-only view or perhaps a cell-only view of the model. Especially with a cell-centric view, it would be interesting to tackle what it means to render a tetrahedron rather than a triangle as the basic modelling unit.

5.3 Rotation

While the order of rotation is satisfying, it is not inherited from any other work. As well, there are limitations to the rotation order, such as once the object has a non-3D rotation applied, the recognizable 3D rotations begin to behave differently. This could be improved by changing the order of rotations to make the 3D rotations independent from the w-axis, however may have other consequences once implemented.

Notably, while this work was implemented using Euler Angles, it should be possible to implement this work with the 4D equivalent of quaternions. This would help with gimbal lock, which was a problem with Tiny's table and also the reason why the pitch and yaw of the viewing plane were locked to not exceed the y-axis. Notably, gimbal lock was not observed in the rotation of the viewing space, which most likely means that it is harder to observe with more degrees of freedom.

It would also help greatly to create a 4D transformation scheme from the ground up, as in the code, the transform matrix was built to be dependent on the editor's euler angles, which caused many separation of power issues as well as optimization issues. Along with Godot's limitation of not having a native matrix class, it would improve the logical flow significantly to not use Godot.

5.4 Projection

Whilst not necessarily a limitation, it should be noted that this method of projection causes the position and rotation of the user's camera affect where objects are and how transparent they are. This makes it difficult to observe objects at certain rotations of the viewing space, as they could end up very far from their actual location, and approaching the object will change it's view.

One potential improvement would be to allow the user to 'freeze' their camera position, so that they can approach the object. Another could be to allow the user to manipulate an external camera, however this was not implemented in this project as one of the motivations was to explore a first person perspective of viewing 4D objects.

5.5 4D Visualization Method

In this method, transparency was used to display information in the w-axis. This caused it to be difficult to tell if a point on the shape was in the positive or negative direction. One possible improvement would be to use color to indicate positive or negative, however if this is implemented I would not recommend a hard boundary between positive and negative, as it was used for debugging, but looked bad. Another potential solution would be to display only one direction in the w-axis, not dissimilar to how a frustum only faces in one direction.

On the topic of w-axis visualization, this method uses a fixed distance from the camera, 1 meter in the 4D visualization, to display objects in the w-axis. However one could imagine systems more similar to a frustum where as distance from the camera increases, objects could be seen from a wider range of the w-axis. This was

attempted, but it caused difficulties observing objects close to the camera.

While the transparency worked well to visualize w-axis information, this project has a major limitation in that the game engine, Godot, cannot correctly distance sort transparent objects. This causes errors such as the w-axis on the four-color axes to always be rendered in front of the hypercube that it was inside of. The main takeaway for future work here is do not use the Godot game engine when there is more than one transparent object. Thankfully the effect on this project is rather minimal as this was most visible on the four-color axes and little elsewhere.

REFERENCES

[1] Edwin Abbott Abbott. 1991. *Flatland: A Romance of Many Dimensions.* Princeton University Press, Princeton, NJ. http://www.jstor.org/stable/j.ctt1d2dmff

[2] David Cotton Banks. 1993. *Interacting with surfaces in four dimensions using computer graphics.* Ph. D. Dissertation. The University of North Carolina at Chapel Hill. http://login.ezproxy.lib.umn.edu/login?url=https://www.proquest.com/dissertations-theses/interacting-with-surfaces-four-dimensions-using/docview/304058904/se-2

[3] Alan Chu, Chi-Wing Fu, Andrew Hanson, and Pheng-Ann Heng. 2009. GL4D: A GPU-based Architecture for Interactive 4D Visualization. *IEEE Transactions on Visualization and Computer Graphics* 15, 6 (Nov 2009), 1587–1594. https://doi.org/10.1109/TVCG.2009.147

[4] CodeParade. 2024. 4D Golf. Steam. https://store.steampowered.com/app/2147950/4D_Golf/

[5] Jonny Collins, Holger Regenbrecht, and Tobias Langlotz. 2021. Expertise and Experience in VR-supported learning: Achieving a deep non-verbal comprehension of four-dimensional space. *International Journal of Human-Computer Studies* 152 (2021), 1–12. https://doi.org/10.1016/j.ijhcs.2021.102649

[6] evgen52. 2023. Magic Cube 4D VR. Steam. https://store.steampowered.com/app/2413000/Magic_Cube_4D_VR/

[7] Christoph M. Hoffman and Jianhua Zhou. 1990. *Visualization of Surfaces in Four-Dimensional Space.* Technical Report TR-960. Purdue University, West Lafayette, IN. https://docs.lib.purdue.edu/cstech/814

[8] H. Igarashi and H. Sawada. 2021. Touching 4D Objects with 3D Tactile Feedback. In *2021 14th International Conference on Human System Interaction (HSI).* IEEE, Gdańsk, Poland, 1–7. https://doi.org/10.1109/HSI52170.2021.9538790

[9] Mashpoe. 2022. 4D Miner. Steam. https://mashpoe.com/4d-miner

[10] David H Laidlaw Prabhat, Thomas F Banchoff, and Cullen D Jackson. 2001. *Comparative Evaluation of Desktop and Cave Environments for Learning Hypercube Rotations.* Master's thesis. Brown University. http://128.148.32.110/research/pubs/theses/masters/2001/prabhat.pdf

[11] Marc ten Bosch. 2017. 4D Toys. Steam. https://4dtoys.com/

[12] Hui Zhang. 2008. *Physically Interacting with four Dimensions.* Ph. D. Dissertation. Indiana University. https://scholarworks.iu.edu/dspace/bitstream/2022/8477/1/Zhang_indiana_0093A_10088.pdf

Received 29 April 2024