Mathematics IA

Visualizing 4D Polygons

How can 4D shapes be visualized by humans?

1 Introduction and Rationale

4D is an elusive concept popularized in many aspects of human life; movies, books, and people all seem to talk about 4D in some way. But what are 4D shapes, and how can humans perceive them?

As a computer graphics enthusiast, I have made many different types of physics simulations. All of which have been constrained to 3D space and lower. As I moved from 2D to 3D, the complexity of geometric interactions increased, and I observed interesting phenomena regarding the change in spatial volume. These observations raised an important question: what happens if I go beyond 3D space? To satisfy my curiosity, I investigated how the visualization of 4D shapes work. Moreover, creating a complete rendering engine is also a long-term goal of mine. I believe this IA topic will be monumental in helping me improve my mathematical abilities in computer graphics. Therefore, as an investigation of 4D visualization, the goal of this IA is to find a transformation matrix that can project any 4D wireframe into human-perceivable 3D space. (Note: all diagrams in this IA are self-created by the author)

2 Human Perception and Perspective Projections

To visualize 4D, it is meaningful to discuss how humans perceive objects in general. In reality, humans actually cannot see 3D space directly. Instead, humans only see a 2D projection of light onto their retinas. The sense of 3D in human sight is an illusion caused by shadows, lights, parallax with two eyes, and clever brain trickery. Fundamentally, the eye can be simplified into a few key components: the lens, the focal point, and the retina. The retina is the 2D plane to which the light will be projected, and the focal point is the pivot that gives the 3D object perspective.

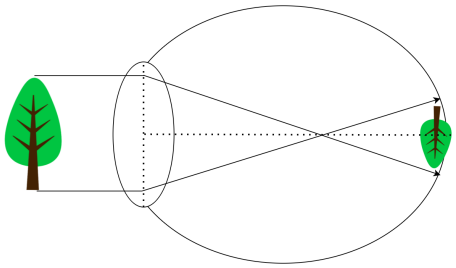


Figure 1: Diagram of the human eye

For this IA, the projection will use the same principles as the eye diagram, with the exception that the retina at the end is flat (Figure 2). The approach taken is closer to that of a pinhole camera.

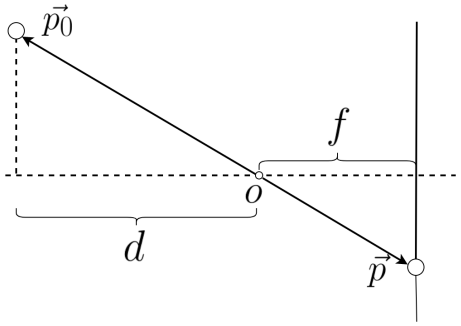


Figure 2: Diagram of the simplified projection system of the eye.

In the diagram, $\vec{p_0}$ is the position vector of the original point, $\vec{p}$ is the position vector of the projected point, and o is the origin. The scalars d and f represent the distance to the focal point and the focal distance respectively. However, the projected object is inverted, and inversion is not ideal because it is not the "real" orientation of the original object. To simplify the projection process, the "retina" (the 2D plane) will be moved to the front of the focal point instead. This still results in a perspective projection because the projection is still scaled by the distance between the original object and the focal point.

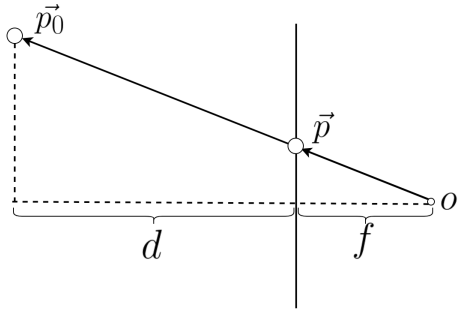


Figure 3: Diagram of the projection system used for this IA

This will form the basis of perspective projections, and the next sections will formalize this result with mathematics.

3 Definitions and Notation

The 4th dimensional axis will be denoted as w. Components of a vector $\vec{v_1}$, will be referred as $v_{1,n}$. To illustrate, the x-component of the $\vec{v_1}$ will be $v_{1,x}$. Matrices will be represented as capitalized letters with a bold font, ie, $\mathbf{A}$. The vectors for this IA will be slightly different from typical spacial vectors. First of all, vectors will be in row form. Meaning, $\vec{v} = x\vec{i} + y\vec{j} + z\vec{k} = [x \quad y \quad z]$. Hence, the vector basis will also be in row form, where for 3D $\vec{i} = [1 \quad 0 \quad 0]$, $\vec{j} = [0 \quad 1 \quad 0]$, $\vec{k} = [0 \quad 0 \quad 1]$. Similarly, the vector basis in 4D will be represented as $\vec{i} = [1 \quad 0 \quad 0 \quad 0]$, $\vec{j} = [0 \quad 1 \quad 0 \quad 0]$, $\vec{k} = [0 \quad 0 \quad 1 \quad 0]$, and $\vec{l} = [0 \quad 0 \quad 0 \quad 1]$. This form was specifically chosen because it is easier to implement programmatically. Because of this, the order of vector and matrix multiplication will be reversed. In other words, multiplication will be in the order of $\vec{v}\mathbf{A}$. Moreover, an additional dimension will be added to all vectors. Therefore, a 3D vector will be represented as $[x \quad y \quad z \quad 1]$, and 4D as $[x \quad y \quad z \quad w \quad 1]$. The additional 1 at the end of each vector will enable translation matrices, $\mathbf{D}$, where $\vec{v}\mathbf{D} = \vec{v'}$. Vectors written in this form have homogeneous coordinates, and any transformations on homogeneous coordinates are called affine transformations (Strang). As a result, 3D spacial vectors will have 4 components, and 3D transformation matrices will be 4×4. Similarly, 4D spacial vectors will have 5 components, and 4D transformation matrices will be 5×5.

4 3D Perspective Projections

Because it is not possible for a human to directly visualize 4D space, a projection process is first developed for 3D space and later extended to 4D. Having an understanding of 3D perspective projections to a 2D space will allow the development of some key aspects of the entire process. This section will discuss how to project 3D points onto 2D space, as points (vertices) are a key component of polygons.

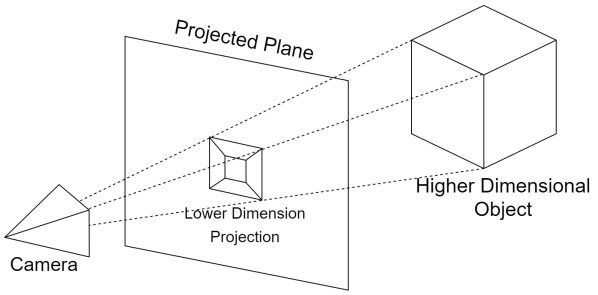


Figure 4: The general 3D projection process

4.1 Simple Perspective Projection Case

This section will be guided by the following diagram

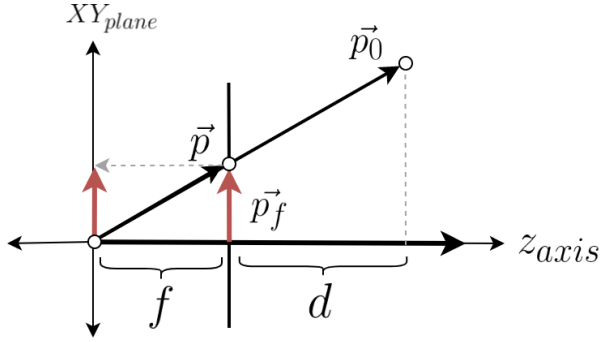


Figure 5: Simple 3D projection

The key to perspective projections is to find the value of $\vec{p_f}$, given $\vec{p_0}$ and f. The final projection $\vec{p_f}$ is in the 2D plane of which it is projected to. To solve for this value, $\vec{p}$ is first calculated using similar triangles. Because the triangle of $\vec{p_0}$ shares two angles as the triangle of $\vec{p}$, they are similar triangles. Therefore,

$$\frac{|\vec{p_0}|}{f+d} = \frac{|\vec{p}|}{f}$$

Hence, isolating $\vec{p}$ yields

$$|\vec{p}| = \frac{f}{f+d}|\vec{p_0}|$$

Because $\vec{p}$ and $\vec{p_0}$ are collinear, $\vec{p}$ must be a scalar multiple of $\vec{p_0}$, and the scalar value must also be same as the scaling for the length. Furthermore, the value $f+d$ is simply just the z-coordinate of the point $\vec{p_0}$. Which results in,

$$\vec{p} = \frac{f}{p_{0,z}}\vec{p_0} \tag{1}$$

This scaling on $\vec{p_0}$ can be rewritten as a scaling matrix, $\mathbf{S}$, that applies the scalar multiple to each component of $\vec{p_0}$. Hence,

$$\vec{p} = \vec{p_0} \begin{bmatrix} \frac{f}{p_{0,z}} & 0 & 0 & 0 \\ 0 & \frac{f}{p_{0,z}} & 0 & 0 \\ 0 & 0 & \frac{f}{p_{0,z}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, to get the position vector $\vec{p_f}$ in the plane of projection (the 2D plane), $\vec{p}$ needs to be translated along the z-axis. The translation will displace the same length as the focal distance, f, and because it is along the z-axis, only the z-component will be affected. To keep everything consistent, a 4×4 translation matrix will be used.

$$\vec{p_f} = \vec{p} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -f & 1 \end{bmatrix}$$

This is the reason why homogeneous coordinates were used, since expanding this translation gives $\vec{p_f} = [p_x \quad p_y \quad (p_z - f) \quad 1]$. Combining the translation and projection results in the final projection matrix, $\mathbf{J}$. Where,

$$\mathbf{J} = \begin{bmatrix} \frac{f}{p_{0,z}} & 0 & 0 & 0 \\ 0 & \frac{f}{p_{0,z}} & 0 & 0 \\ 0 & 0 & \frac{f}{p_{0,z}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -f & 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{p_{0,z}} & 0 & 0 & 0 \\ 0 & \frac{f}{p_{0,z}} & 0 & 0 \\ 0 & 0 & \frac{f}{p_{0,z}} & 0 \\ 0 & 0 & -f & 1 \end{bmatrix} \tag{2}$$

and $\vec{p_f} = \vec{p_0}\mathbf{J}$.

4.2 Translated Camera Coordinates

The camera coordinates do not always need to be at the origin, and they could be translated in space. The following diagram will illustrate the scenario for translated camera coordinates.

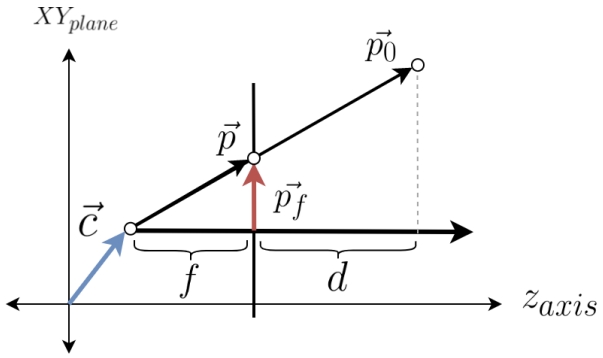


Figure 6: 3D projection with translated camera coordinates

This issue can be resolved by simply applying a counter translation such that the camera moves back to the origin. The system can then be projected normally, just like in Section 4.1. Let the camera be represented by the position vector $\vec{c} = [c_x \quad c_y \quad c_z \quad 1]$. Then, the counter translation matrix, $\mathbf{D_c}$, will be,

$$\mathbf{D_c} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -c_x & -c_y & -c_z & 1 \end{bmatrix} \tag{3}$$

Such that, $\vec{c}\,\mathbf{D_c} = [(c_x - c_x) \quad (c_y - c_y) \quad (c_z - c_z) \quad 1] = 0$. (The 0 vector also has homogeneous coordinates)

4.3 Rotated Camera Angle

Just like how the camera coordinates can change, the angle of the camera can also change accordingly. The following diagram will outline the case where the normal vector of the projection plane does not lie on the z-axis.

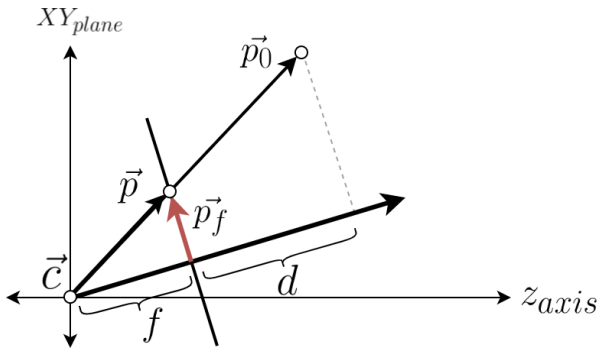


Figure 7: 3D projection with rotated camera direction

This system can be solved by counter-rotating, such that the axes of the camera system move back onto the spacial axes. Although the direction of the camera can be directly determined by the normal vector of the projection plane, doing so will make camera angles unclear. Therefore, the direction is alternatively represented by a set of angles. As such, the goal state of the system will be where all the angles are 0. In 3D space, any direction vector can be determined with 3 angles α, β, and γ (Figure 8).

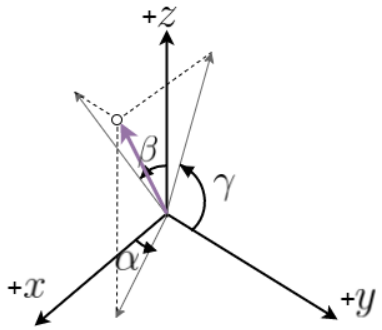


Figure 8: Diagram of how α, β, and γ are used to determine the direction in 3D space

The change in each of those angles will coincide with a rotation about a different axis. Similar to the case with translated camera coordinates, a counter rotation will be done for each angle such that it returns in the simple case. These rotations will be done through a series rotation matrices, $\mathbf{R_x}$, $\mathbf{R_y}$, and $\mathbf{R_z}$, for a rotation about each axis. In order to form a generalized method used to derive the rotation matrices, it is important to understand the principles of how transformation matrices affect a vector. So, for a transformation matrix, $\mathbf{T} = [\vec{r_1} \quad | \quad \vec{r_2} \quad | \quad \vec{r_3} \quad | \quad \vec{r_4}]^\intercal$ ($\vec{r_i}$ are the rows vectors of T), applied on a 3D vector $\vec{v} = [x \quad y \quad z \quad 1]$,

$$\vec{v}\,\mathbf{T} = x\vec{r_1} + y\vec{r_2} + z\vec{r_3} + 1\vec{r_4}.$$

The value of $\vec{r_4}$ should just be the original basis vector, l, as the transformation should not effect the 1 at the end (it is only there to enable affine transformations). Upon closer inspection, this is the same effect as applying the transformation individually on each vector

basis of the original space, where

$$\vec{v}\,\mathbf{T} = x(\vec{i}\,\mathbf{T}) + y(\vec{j}\,\mathbf{T}) + z(\vec{k}\,\mathbf{T}) + 1\vec{l} = x\vec{r_1} + y\vec{r_2} + z\vec{r_3} + 1\vec{l} \tag{4}$$

Hence, to find the rotation matrix, the effects on each vector basis need to be individually calculated and joined together later. For example, the rotation matrix $\mathbf{R_z}$ will result in a rotation about the z-axis which affects the x and y values, or the $\vec{i}$ and the $\vec{j}$ basis.

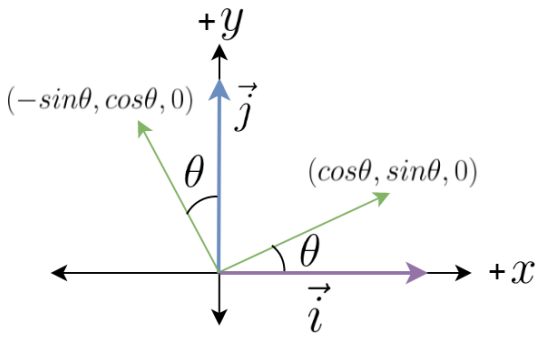


Figure 9: Example of rotation about the z-axis (the z-axis is coming out of the page)

Using some simple trigonometry, the rotated coordinates of each basis can be calculated. Hence, the position vectors of the rotated $\vec{i}$ is $[cos\theta \quad sin\theta \quad 0 \quad 0]$ and the rotated $\vec{j}$ is $[-sin\theta \quad cos\theta \quad 0 \quad 0]$ (the new position vectors do not have homogeneous coordinates because the basis vectors do not have homogeneous coordinates either). The remaining vector bases, $\vec{k}$ and $\vec{l}$, are unchanged because the rotation does not influence them. Hence, the combined rotation matrix will be

$$\mathbf{R_z} = \begin{bmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because the z-axis rotation is characterized by the angle α, the counter rotation will have angles $-\alpha$. Therefore,

$$\mathbf{R_z} = \begin{bmatrix} cos(-\alpha) & sin(-\alpha) & 0 & 0 \\ -sin(-\alpha) & cos(-\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Using the same method for R_x (affects $\vec{j}$ and $\vec{k}$), and R_y (affects $\vec{i}$ and $\vec{k}$), will give

$$\mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) & 0 \\ 0 & sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R_y} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{R_z} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

4.4 Combined Process

The key of the combined process is to simplify the system such that it becomes the simple case. Therefore, the steps to this process will be

1. Translate system such that the camera is at the origin (Section 4.2)
2. Rotate system such that camera angles are all 0 (Section 4.3)
3. Project the points (Section 4.1)

This combination is achieved by multiplying all the different matrices together in that order (the ease of combination is also the reason why matrices were chosen).

$$\vec{p_f} = \vec{p_0}\mathbf{D_c}\mathbf{R_x}\mathbf{R_y}\mathbf{R_z}\mathbf{J},$$

where $\mathbf{D_c}$ is the translation matrix, $\mathbf{R_x}$, $\mathbf{R_y}$, and $\mathbf{R_z}$ are the rotation matrices, and $\mathbf{J}$ is the projection matrix. However, the projection matrix will require values of $\vec{p_0}\mathbf{D_c}\mathbf{R_x}\mathbf{R_y}\mathbf{R_z}$, as it needs the resulting w-component of the transformed vector to find the perspective scaling. Therefore, the projection matrix, J is rewritten as,

$$J_T = \begin{bmatrix} \frac{f}{p_{T,z}} & 0 & 0 & 0 \\ 0 & \frac{f}{p_{T,z}} & 0 & 0 \\ 0 & 0 & \frac{f}{p_{T,z}} & 0 \\ 0 & 0 & -f & 1 \end{bmatrix},$$

where $\vec{p_T} = \vec{p_0}\mathbf{D_c}\mathbf{R_x}\mathbf{R_y}\mathbf{R_z}$. The final equation for 3D perspective projections will then be

$$\vec{p_f} = \vec{p_0}\mathbf{D_c}\mathbf{R_x}\mathbf{R_y}\mathbf{R_z}\mathbf{J_T}. \tag{6}$$

These matrices will not be condensed because it does not simplify the equation: it just makes it more convoluted.

5 4D Perspective Projections

After establishing an understanding of 3D perspective projections, the IA can now move on to 4D perspective projections. Because it is impossible to visualize 4D space directly, a lot of the explanations will use analogies from lower dimensions. Moreover, the pre-established mathematical understanding from 3D projection can also be extended towards 4D projections. This section will discuss how to project 4D points onto 3D space, as points (vertices) are a key component of understanding 4D polygons.

5.1 4D Projections

The entire perspective projection method will remain the same, with the exception that the depth is now caused by is w-axis. Hence, the translation at the end of the projection will be on the w-component of $\vec{p_0}$. This also holds true for any point $\vec{p_n}$, as long its a position vector. Hence, the general projection matrix for $\vec{p_n}$ is,

$$\mathbf{J_n} = \begin{bmatrix} \frac{f}{p_{n,w}} & 0 & 0 & 0 & 0 \\ 0 & \frac{f}{p_{n,w}} & 0 & 0 & 0 \\ 0 & 0 & \frac{f}{p_{n,w}} & 0 & 0 \\ 0 & 0 & 0 & \frac{f}{p_{n,w}} & 0 \\ 0 & 0 & 0 & -f & 1 \end{bmatrix} \tag{7}$$

such that $\vec{p_f} = \vec{p_n}\mathbf{J_n}$ and $\vec{p_f}$ is the projected vector. When calculated correctly, $p_{f,w} = p_{n,w}\left(\frac{f}{p_{n,w}}\right) - f = 0$, which verifies that the final projected vector is in lower dimensional space.

5.2 4D Translations

Just like 3D translations, 4D translation will move the camera back to the origin. The only exception is the extra dimension. Hence, the camera will will have a position vector

$\vec{c} = [c_x \quad c_y \quad c_z \quad c_w \quad 1]$. Therefore, the counter translation matrix will be,

$$\mathbf{D_c} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -c_x & -c_y & -c_z & -c_w & 1 \end{bmatrix} \tag{8}$$

5.3 4D Rotations

The process to solve for rotated camera angles in 4D is similar to that of 3D: find a counter-rotation matrix that can rotate the camera back into standard position. Although 4D rotations share similarities with 3D rotations, it still has distinct differences that need to be addressed. Because of their increased degree of freedom, 4D objects rotate through space differently than 3D. 4D objects rotate about planes, not lines. This concept is similar to how a 2D shape only rotates about a point. A rotation through a new spatial dimension also requires a spatial extension of the rotation center. To understand this better, rotations need to be visualized differently. Instead of thinking of rotations as movement around a stationary center, it is better to visualize rotations as a planar movement. To illustrate, a 2D rotation revolves around an object in a plane, but since there is only one 2D plane available in 2D space, the stationary center will be a point of 0 dimensions. Similarly, rotations in 3D also occur on a rotational plane, but since 3D has one extra dimension, the resulting stationary center will be a line. Lastly, rotations in 4D are also in a 2D rotational plane, so the resulting stationary points must be the leftover 2 dimensions, which is a plane. Because of this, there will be ${}_4C_2 = 6$ different rotations in 4D space. To define these 6 rotations, 6 distinct angles, α, β, γ, δ, ϵ, and ζ will be used. Each of these angles will pertain to a rotation matrix, $\mathbf{R_{xy}}$, $\mathbf{R_{yz}}$, $\mathbf{R_{xz}}$,$\mathbf{R_{xw}}$, $\mathbf{R_{yw}}$, and $\mathbf{R_{zw}}$. Since it is not possible to draw 4D axes, they will instead be broken down into three different 3D spatial axes. This works because all of the 4D axes are orthogonal to each other.

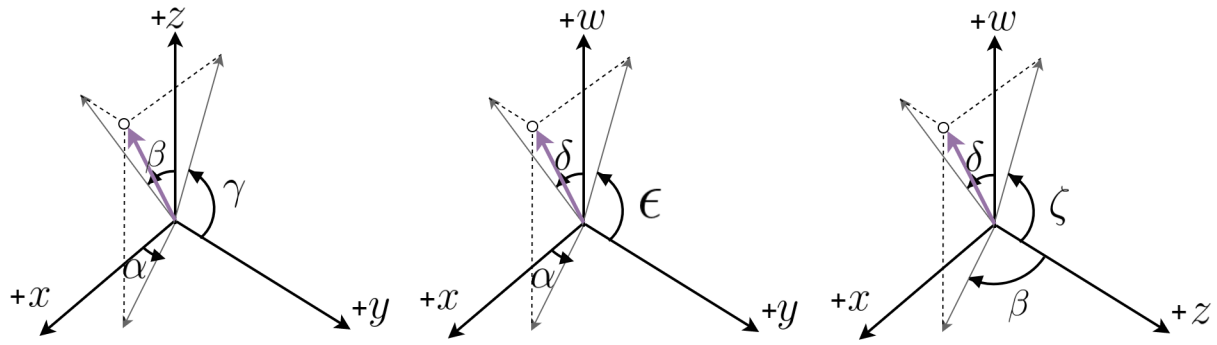


Figure 10: Dissection of rotation angles for a 4D vector (diagram is not to scale nor geometrically accurate)

The derivation of the rotation matrices will remain the same as the 3D variation, where the transformation is just applied to the vector basis. But for 4D, there will be one extra vector basis and one additional dimension. For example, $\mathbf{R_{xy}}$ is the rotation about the XY plane, which is a rotation in the ZW plane.

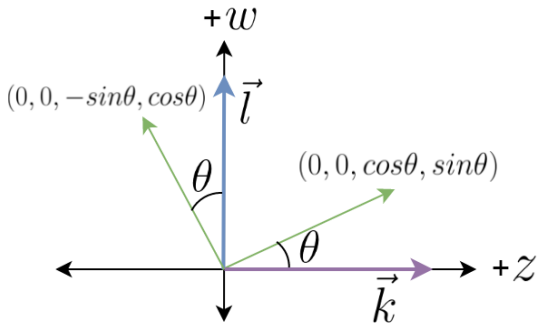


Figure 11: Example of 4D rotation about the XY plane

From the above diagram, the rotated value of $\vec{k}$ is $[0 \quad 0 \quad cos\theta \quad sin\theta \quad 0]$, and the rotated value of $\vec{l}$ is $[0 \quad 0 \quad -sin\theta \quad cos\theta \quad 0]$. Moreover, since $\mathbf{R_{xy}}$ causes a rotation in the ZW plane, the corresponding angle is ζ, and the counter rotation angle will be $-\zeta$. Hence, combining all the transformed basis vectors will yield the counter rotation matrix,

$$\mathbf{R_{xy}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & cos(-\zeta) & sin(-\zeta) & 0 \\ 0 & 0 & -sin(-\zeta) & cos(-\zeta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & cos(\zeta) & -sin(\zeta) & 0 \\ 0 & 0 & sin(\zeta) & cos(\zeta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Applying the same process for $\mathbf{R_{yz}}$ (Rotation on XW plane; angle δ), $\mathbf{R_{xz}}$ (Rotation on YW plane; angle ϵ), $\mathbf{R_{xw}}$ (Rotation on YZ plane; angle γ), $\mathbf{R_{yw}}$ (Rotation on XZ plane; angle

β), and $\mathbf{R_{zw}}$ (Rotation on XY plane; angle α) gives the set of counter rotation matrices:

$$\mathbf{R_{xy}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & cos(\zeta) & -sin(\zeta) & 0 \\ 0 & 0 & sin(\zeta) & cos(\zeta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R_{xz}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & cos(\epsilon) & 0 & -sin(\epsilon) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & sin(\epsilon) & 0 & cos(\epsilon) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_{yz}} = \begin{bmatrix} cos(\delta) & 0 & 0 & sin(\delta) & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -sin(\delta) & 0 & 0 & cos(\delta) & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R_{xw}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & cos(\gamma) & -sin(\gamma) & 0 & 0 \\ 0 & sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$\mathbf{R_{yw}} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R_{zw}} = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 & 0 & 0 \\ sin(\alpha) & cos(\alpha) & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

5.4 Combined Projection Process

The combined projection process in 4D will also move the system to the simple projection case. Therefore, the order of the process is the same as the 3D version:

1. Translate system such that the camera is at the origin (Section 5.2)

2. Rotate system such that camera angles are all 0 (Section 5.3)

3. Project the points (Section 5.1)

Given a camera with position vector $\vec{c} = [c_x \quad c_y \quad c_z \quad c_w \quad 1]$, defined by the angles α, β, γ, δ, ϵ, and ζ. The combined transformation on a 4D vector $\vec{p_0}$ gives the general equations:

$$\begin{aligned} \vec{p_T} &= \vec{p_0}\mathbf{D_c}\mathbf{R_{xy}}\mathbf{R_{xz}}\mathbf{R_{yz}}\mathbf{R_{xw}}\mathbf{R_{yw}}\mathbf{R_{zw}} \\ \vec{p_f} &= \vec{p_0}\mathbf{D_c}\mathbf{R_{xy}}\mathbf{R_{xz}}\mathbf{R_{yz}}\mathbf{R_{xw}}\mathbf{R_{yw}}\mathbf{R_{zw}}\mathbf{J_T} = \vec{p_T}\mathbf{J_T}. \end{aligned} \tag{10}$$

Where $\vec{p_T}$ is the resultant from the matrix transformations, $\vec{p_f}$ is the resulting projection, $\mathbf{D_c}$ is the translation matrix, $\mathbf{R_i}$ are the 4D rotation matrices, and $\mathbf{J_T}$ is the projection matrix of $\vec{p_T}$. Note that the resulting vector, $\vec{p_f}$, should be in the form of $[p_{f,x} \quad p_{f,y} \quad p_{f,z} \quad 0 \quad 1]$. The w-component, $p_{f,w}$, is 0, which shows that the resulting projection is indeed in lower

dimensional space. Therefore, the final vector with reduced dimensions for rendering can be rewritten as $\vec{p_f}' = [p_{f,x} \quad p_{f,y} \quad p_{f,z}]$ (homogeneous coordinates are removed, they are not required for rendering).

5.4.1 Dealing With Edges

Edges are a crucial part of visualizing polygons. An edge is defined as a line connecting a pair of points. Because an edge is a line with no thickness, its projection process is relatively simple. Instead of dealing with additional calculations, the projected edge in 3D can be reconstructed with the two known projected vertices. The relationship of an edge will remain the same despite projection; the edge connecting p_1 and p_2 will also be true after projection. Hence, a simple line is just drawn between the two corresponding projected points.

6 Programmatic Implementation

To verify the results, the general equation is implemented programmatically in JavaScript. The following algorithm will outline the general process.

Algorithm 1 4D Perspective Projection Implementation

$\mathbf{D_c} = computeTranslationMatrix()$
$\mathbf{R_{xy}}$, $\mathbf{R_{xz}}$, $\mathbf{R_{yz}}$, $\mathbf{R_{xw}}$, $\mathbf{R_{yw}}$, $\mathbf{R_{zw}} \leftarrow computeRotationMatrices()$
for all $p_i \in \mathbb{R}^4$ **do**
    $\vec{p_T} = \vec{p_0}\mathbf{D_c}\mathbf{R_{xy}}\mathbf{R_{xz}}\mathbf{R_{yz}}\mathbf{R_{xw}}\mathbf{R_{yw}}\mathbf{R_{zw}}$
    $\mathbf{J_T} \leftarrow computeProjectionMatrix(p_T)$
    $\vec{p_f} = \vec{p_T}\mathbf{J_T}$
    $Draw\ p_f$
end for
for all $edge_i \in \mathbb{R}^4$ **do**
    // an edge is simply a relationship between 2 points, this relationship is retained during projections
    $Draw\ edge_i$
end for

A 4D object in the implementation will consist of a set of 4D points and a set of edges that connect certain points together. To aid the evaluation, a popular 4D shape, the tesseract, is used during the 4D perspective projection process. A tesseract is the 4D equivalent of a

cube, and for this particular implementation, it will have a side length of 200 and the center at the origin. The position the 4D camera will be defined using homogeneous coordinates as $\vec{c} = [0 \quad 0 \quad 0 \quad 90 \quad 1]$. The focal distance, f, will be 100. Additionally, the angles α, β, γ, δ, ϵ, and ζ are all initially 0. In the 3D view, the center of the page is the origin, while the left-right direction corresponds to the x coordinates, the up-down direction to the y coordinates, and the depth to the z-coordinates. Access to the implementation is available in the Appendix.

7 Evaluation and Reflection

The initially loaded projection of the tesseract is given below. The results are similar to projections outlined in Helmenstine's article.

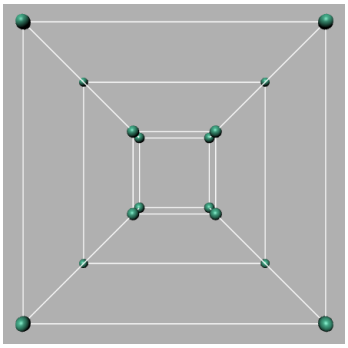


Figure 12: Initial perspective projection of the tesseract

Next, the initial parameters will be altered to evaluate how the projection behaves. First, the angle parameters will be altered. Each angle corresponds to a rotation of the camera direction, and it yields different projections. From experimentation, camera rotations can be generalized into 2 categories: rotations that do not affect w-component ($\mathbf{R_{xw}}$, $\mathbf{R_{yw}}$, $\mathbf{R_{zw}}$) and rotations that do affect the w-component ($\mathbf{R_{xy}}$, $\mathbf{R_{yz}}$, $\mathbf{R_{xz}}$). Rotations that do not affect the w-component exhibit clear rotation behaviour:

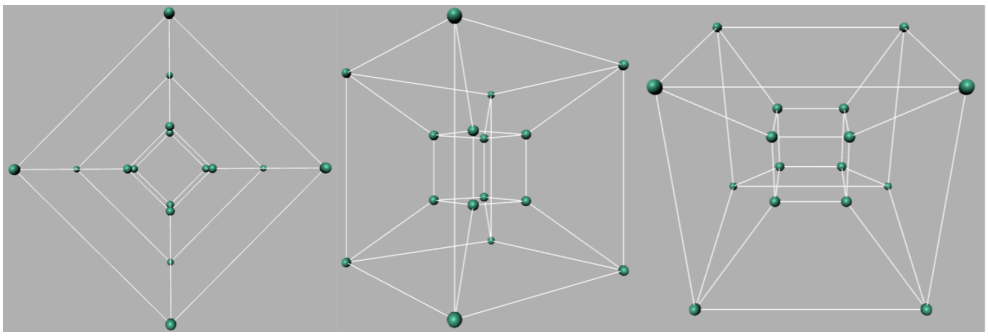


Figure 13: Projection as a result of $\mathbf{R_{xw}}$, $\mathbf{R_{yw}}$, $\mathbf{R_{zw}}$ (angles are in radians)

Upon closer inspection, the results are identical to directly rotating the 3D projection. Intuitively, this makes sense because each of those angles corresponds to a rotation about a 2D plane. For example, $\mathbf{R_{zw}}$ results in a rotation in the XY plane, characterized by the angle α. Moreover, because the perspective from the 4D projection is caused by the w component, $\mathbf{R_{zw}}$ will not affect the perspective of the final project. Therefore, changing α will simply cause a rotation in the XY plane about the z-axis for the 3D projection. To put mathematically,

$$\vec{p_0}\mathbf{R_{nw}} = \vec{p_f}\mathbf{R_n},$$

where n is the axis for the plane NW of which $\mathbf{R_{nw}}$ revolves around, and $\mathbf{R_n}$ is the 3D rotation matrix that describes a rotation about the n-axis.

However, rotations that affect the w-component are much more complex and cause the projection to warp. Just to reiterate, all rotations discussed in this section pertain to the rotation of the camera direction, not of the objects themselves. Rotations that affect the w-component seem to skew the projection:

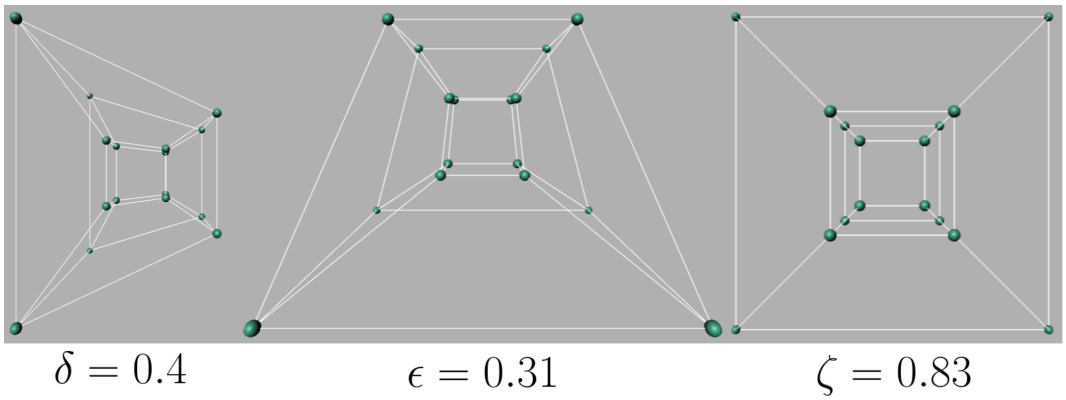


Figure 14: Projection as a result of $\mathbf{R_{xy}}$, $\mathbf{R_{yz}}$, $\mathbf{R_{xz}}$ (angles are in radians)

The skew and scaling effects imply the existence of angle boundaries or else the scaling will be infinite. The maximum angle depends on the position of $\vec{p_0}$ and will occur when the rotation causes the 3D projection space to intersect with $\vec{p_0}$ in 4D. Turning any further from this angle will cause the camera view to exclude the 4D space of the object in it. To help visualize this explanation, a 3D case is drawn below:

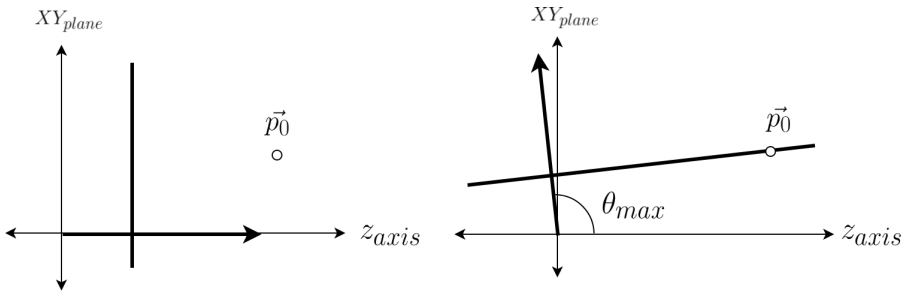


Figure 15: Diagram of the maximum rotation angle (θ_{max}) for a point p_0

Moving on, each of the camera coordinates, c_x, c_y, c_z, and c_w, will be modified. The last coordinate, w is a special case as it directly affects the 4D depth and the sense of perspective of the 3D projection. The rest of the coordinates only shift and scale the projection.

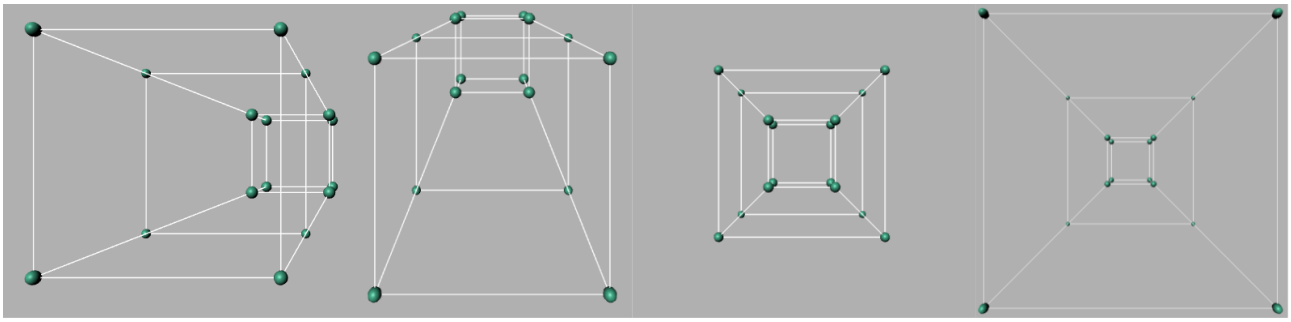


Figure 16: Resulting projection from changing camera coordinates (originally $x, y, z = 0$ and $w = -90$)

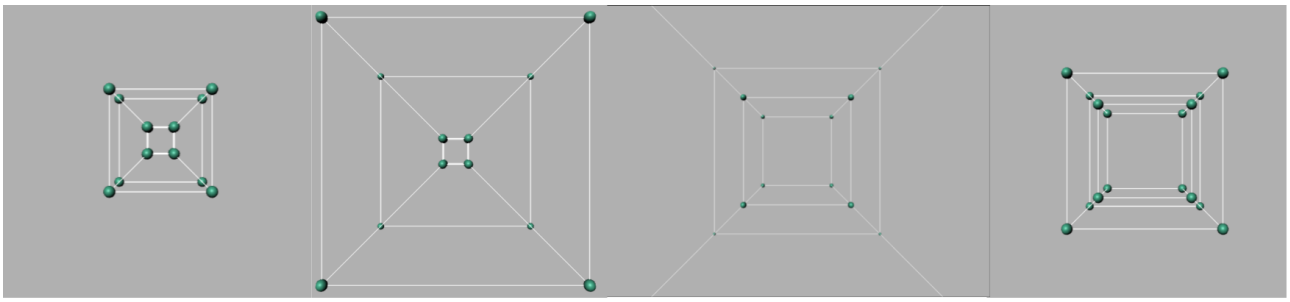
Changing c_x, c_y, and c_z, causes a linear translation and scaling along that axis. This can be demonstrated using the derived 4D projection method. Assuming no other rotations, Equation 10 becomes,

$$\vec{p_f} = \vec{p_0}\mathbf{D_c}\mathbf{J} = \vec{p_0}\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -c_x & -c_y & -c_z & -c_w & 1 \end{bmatrix} \begin{bmatrix} \frac{f}{p_{T,x}} & 0 & 0 & 0 & 0 \\ 0 & \frac{f}{p_{T,y}} & 0 & 0 & 0 \\ 0 & 0 & \frac{f}{p_{T,z}} & 0 & 0 \\ 0 & 0 & 0 & \frac{f}{p_{T,w}} & 0 \\ 0 & 0 & 0 & -f & 1 \end{bmatrix}$$

$$= \vec{p_0}\begin{bmatrix} \frac{f}{p_{T,w}} & 0 & 0 & 0 & 0 \\ 0 & \frac{f}{p_{T,w}} & 0 & 0 & 0 \\ 0 & 0 & \frac{f}{p_{T,w}} & 0 & 0 \\ 0 & 0 & 0 & \frac{f}{p_{T,w}} & 0 \\ -c_x\frac{f}{p_{T,w}} & -c_y\frac{f}{p_{T,w}} & -c_z\frac{f}{p_{T,w}} & -f - c_w\frac{f}{p_{T,w}} & 1 \end{bmatrix}$$

It is evident from the derived matrix that a change in a either c_n, $n \in x, y, z$, will result in $p_{f,n} = \frac{f}{p_{T,w}}(p_{0,n} - c_n)$ (a translation then scaling) where $\vec{p_T} = \vec{p_0}\mathbf{D_c}$. However, a change in c_w differs because it will directly change the value of $\frac{f}{p_{T,0}}$, therefore affecting the degree of scaling for all the projected vectors. Larger $|c_n|$ results in a smaller image, just like how objects look smaller when they are further away.

This leads to the next and last parameter, the focal length f. This parameter, along with c_w, is responsible for the amount of perspective in the projection.



$f = 34, c_w = -90$ $f = 34, c_w = -65$ $f = 247, c_w = -90$ $f = 247, c_w = -230$

Figure 17: Change of perspective resulting from changing f values

The c_w in the above figure only serves to zoom in to or out of the projected image. Decreasing the focal distance, f, changes the perspective by making the perspective effect more pronounced. In other words, closer points are scaled much larger, and further points are scaled smaller. Conversely, increasing f causes the perspective effect to decrease, and points will be scaled less regardless of their distance from the camera. This perspective scaling directly corresponds to the ratio $\frac{f}{p_{T,w}}$. Interestingly, $\lim_{f\to\infty} \frac{f}{p_{T,w}} = \infty$, which seemingly does not result in useful projection results. Understanding this limit requires taking a look at Figure 5. When f is approaching infinity, the perspective triangle will not exist, and there will only be a set of parallel lines. Hence, any projections in this form will not show any perspective, and they are called orthogonal projections (projection is achieved by directly displacing $\vec{p_0}$ by a vector orthogonal to the projected space). On the other side, when f approaches 0, $\lim_{f\to 0} \frac{f}{p_{T,w}} = 0$, and all projections decompose into the origin.

7.1 Limitations and Extensions

Firstly, planes are not considered in the projection, this results in a confusing wire-frame model that cannot fully demonstrate what a 4D polygon looks like. Perhaps an extension similar to the edge treatment (Subsection 5.4.1) could be added to render planes. Moreover, lighting in 4D is also not taken into consideration. Partially because objects are a wireframe model, no lighting can be considered at all. This is a significant limitation as lighting is an important part of computer graphics, and it can also help enhance the comprehensibility of the projected result. Hence, lighting is also a recommended addition in the

future.

Viewport clipping is also not considered in this IA. In real life, the size of the projected space is constrained, just like how a human retina has a set size. Instead, the method in this IA assumes an infinity large projected space with no angle constraints. Sometimes, the implementation encounters erroneous rendering in viewing dead zones. A 4D viewing dead zone is analogously represented in 3D below.

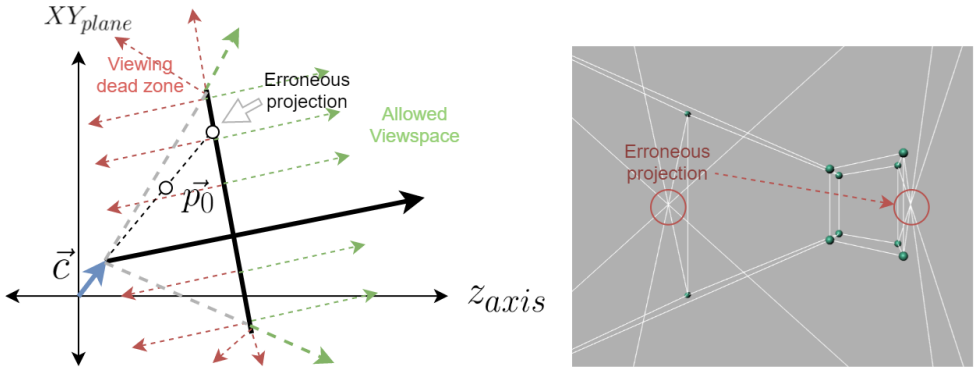


Figure 18: Erroneous rendering of p_0 in visual dead zones (Left: 3D explanation. Right: observed 4D example)

The addition of a viewport clipping algorithm should be considered for the future. Furthermore, FOV (Field of View) is an important concept that this IA does not consider explicitly. FOV provides a more realistic projection method as it is closely related to sight and perspective. Extensions regarding the use of FOV angles in the final projection equation are recommended addition. Lastly, the set of matrices required for perspective projection is quite resource-intensive. Namely, the trigonometric functions in the rotation matrices are expensive to compute. Perhaps a better and more efficient method is available that can avoid the tedious rotation matrices.

8 Conclusion

For this IA, a generalized method was developed to render 4D polygons that only consist of vertices and edges. The final method is mathematically written as a matrix transformation (Equation 10), that takes into account translated and rotated cameras. Additionally, this method was implemented with JavaScript to evaluate its efficacy (Section 6).

Overall, the method behaves as expected and was able to render a tesseract with variable camera parameters. However, the current system has flaws such as the lack of lighting, planar rendering, viewport clipping, and explicit FOV consideration.

Although 4D rendering may seem vague and not obviously useful, it can actually be used in some fields of physics. For example, the crystal structure of quasi-crystals in material science is studied with the help of 4D projections onto 3D space. Perhaps the predominant use of 4D rendering would be its use in understanding space-time. The transformations developed in this IA can be applied to tangible scientific theories, one of them being Lorentz transformations. Moreover, investigating higher dimensions is very mathematically beautiful, and it can enhance the understanding of human observable spatial dimensions. Sometimes, it is worth it to do math just for the sake of doing math, and for the inherent satisfaction and beauty the process brings. I felt that this process has been very helpful in my journey in computer graphics; the mathematics learned is undoubtedly extremely useful for higher-level investigations. Similarly, the development of a basic 4D rendering engine opened plenty of intriguing and artistic paths of investigation for the future.

9 Bibliography

Helmenstine, Anne. "What Is a Tesseract or Hypercube?" Science Notes and Projects, Science Notes and Projects, 19 Dec. 2021, sciencenotes.org/tesseract/. Accessed Jan 1, 2022.

Strang, Gilbert. Introduction to Linear Algebra. Wellesley-Cambridge Press, 2003.

10 Appendix

The implementation demonstration is a website that requires WebGL and javaScript. It will most likely work on any modern browser on a computer.

Implementation: `https://onlinedocumentation.github.io/4dprojection/`

Source Code: `https://github.com/onlineDocumentation/4dprojection`