Summary

The geometry system for 4-dimensional objects in Roblox Studio. The next step is to implement 4-dimensional physics, for which linear and angular motion systems have already been created but not documented here. Collision detection and resolution systems are not yet done.

Geometry

Similarly to how triangles (2-simplex) are used to define meshes in 3 dimensions, tetrahedrons (3-simplex) are used to define meshes in 4 dimensions. The geometry of 4-dimensional objects is stored in five arrays :

- The vertices of the object (4x1 column vectors)
- The edges (as pairs of indices to the vertex array)
- The tetrahedrons (as quadruplets of indices to the vertex array)

Orientation and position are stored in a 5x5 matrix (referred to as a CFrame or coordinate frame) consisting of a 4x4 rotation matrix, 4x1 position vector, and 1x4 identity vector. The CFrame's columns can be interpreted as the right-vector, up-vector, forward-vector, w-vector, and position in world coordinates. The X, Y, Z, and W (right, up, forward, w) vectors together constitute the rotation matrix of the CFrame.

X_x	Y_x	Z_x	W_x	P_x
X_y	Y_y	Z_y	W_y	P_y
X_z	Y_z	Z_z	W_z	P_z
X_w	Y_w	Z_w	W_w	P_w
0	0	0	0	1

CFrame (5x5 matrix). Rotation matrix (4x4). Position (4x1)

The position of any vertex of a shape can be obtained by multiplying its CFrame by the original position of the vertex.

vertexPosition = CFrame * originalVertexPosition

Rotation matrices can be inversed by taking their transpose. Similarly, a CFrame can be inversed by transposing the rotation matrix and setting the position to

$$(-(X \cdot P), -(Y \cdot P), -(Z \cdot P), -(W \cdot P))$$

Rotation matrices

Arbitrary orientations can be generated using the rotation matrices.

R_{XY} =

cosΘ	-sinΘ	0	0
sinΘ	cosΘ	0	0
0	0	1	0
0	0	0	1

R_{XZ} =

cosΘ	0	-sinΘ	0
0	0	0	0
sinΘ	0	cosΘ	0
0	0	0	1

R_{XW} =

cosΘ	0	0	-sinΘ
0	1	0	0
0	0	1	0
sinΘ	0	0	cosΘ

R_{YZ} =

1	0	0	0
0	cosΘ	-sinΘ	0
0	sinΘ	cosΘ	0
0	0	0	1

R_{YW} =

1	0	0	0
0	cosΘ	0	-sinΘ
0	0	1	0
0	sinΘ	0	cosΘ

R_{ZW} =

1	-0	0	0
0	1	0	0
0	0	cosΘ	-sinΘ
0	0	sinΘ	cosΘ

Rendering

In this simulation, 4-dimensional objects are visualized by taking their cross-section or "slice". Slicing a tetrahedron produces 0, 1, or 2 triangles which can be directly rendered. Slicing a tetrahedron is done by individually slicing each edge of the tetrahedron (pre-computed from the vertices and tetrahedron arrays). To do so, write the equation of the edge as a linear interpolation between its two points

$$\text{crossSectionPoint} = \text{vertexA} * (1 - t) + \text{vertexB} * (t)$$

setting w to 0

$$\text{vertexA}_W * (1 - t) + \text{vertexB}_W * (t) = 0$$

which yields

$$t = \text{vertexA}_W / (\text{vertexA}_W - \text{vertexB}_W)$$
$$t \in [0, 1]$$

which we can substitute into the linear interpolation to get the cross-section point. If t is outside of the bounds [0, 1] then the edge does not intersect the 3d hyperplane and does not have a cross-section.