19BIO112

INTELLIGENCE OF BIOLOGICAL SYSTEMS II

TESSERACT

AND NCBI BLAST SEARCH

SIVAMARAN

CB.EN.U4AIE19061

BTECH CSE-AI

CEN DEPARTMENT

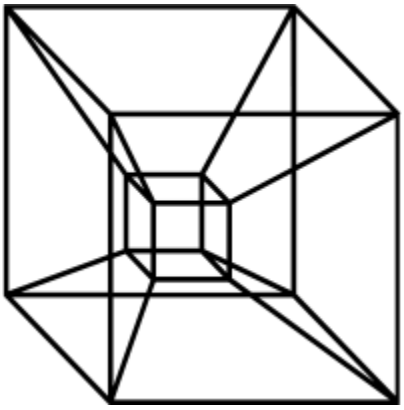
AMRITA SCHOOL OF ENGINEERING

TABLE OF CONTENTS

TESSERACT 3
METHODOLOGY 3
STEREOGRAPHIC PROJECTION 5
WORKING PRINCIPLE 6
PROCESSING CODE IN JAVA 7
OUTPUT OF CODE 10
NCBI BLAST SEARCH 11
TABLE WITH SEARCH RESULTS 11
JUSTIFICATION OF RESULTS 11

PART-1

Animation of Tesseract



What is a Tesseract?

In geometry, the tesseract is the four-dimensional analogue of the cube; the tesseract is to the cube as the cube is to the square. Just as the surface of the cube consists of six square faces, the hypersurface of the tesseract consists of eight cubical cells. The tesseract is one of the six convex regular 4-polytopes.

What is 4^{th} dimension?

As beings living in the three-dimensional world, we are familiar up to 3 dimensions that is length, breath and height. The next question that comes up is what is 4^{th} dimension? Generally speaking, when we talk about a fourth dimension, it's considered space-time. In reality humans are incapable of visualizing in the 4^{th} dimension. That is why such dimensions are mistaken for in popular sci-fi shows. Now we would require a mechanism for visualizing a 4d figure in a 3d world.

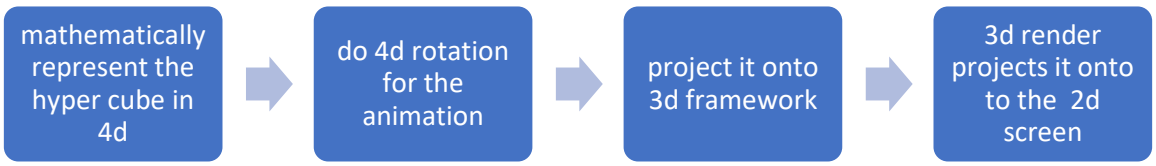
Methodology:

Let’s examine our eyes which are the sensory organs that enable us to perceive the 3d world around us. In fact, you see only two dimensions. This is because what we see is merely 2D images, even of 3D objects, projected onto the backs of

our eyes. What we perceive is entirely different: our brains process those 2D images into something that appears to have not only height and width but also depth – three dimensions.

That's is why our brain easily falls for optical illusions. We would follow a similar mechanism followed by the eye in visualizing the 4d hypercube(tesseract).

We have arrived to the point that from the point of view of an object in 3rd dimension, a 4th dimension does not seem to exist. Therefore, we are physically limited to the 3rd dimension. However, mathematics is not confined to any particular dimension. This is where linear algebra kicks in. In linear algebra a vector of length n represents a quantity in nth dimension, using which we have representation beyond 3rd dimension.



Stereographic projection

In the above flowchart we have done projection from a dimension to a lower dimension. In this regard projections remain conceptually powerful as they are widely used in machine learning algorithms and principal component analysis. In our case we have used stereographic projection.

In geometry, the stereographic projection is a particular mapping (function) that projects a sphere onto a plane. The projection is defined on the entire sphere, except at one point: the projection point. Intuitively, then, the stereographic projection is a way of picturing the sphere as the plane, with some inevitable compromises. Because the sphere and the plane appear in many areas of mathematics and its applications, so does the stereographic projection.

In cartesian coordinates (x, y, z)

$$(X, Y) = \left(\frac{x}{1-z}, \frac{y}{1-z}\right)$$

Generalization:

$$X_i = \frac{x_i}{1-x_0} \quad (i \text{ from } 1 \text{ to } n)$$

Working

Stage 1:

Create the 16 vertices of the hyper cube in the 4^{th} dimension using P4vector class and store it in a vector. Draw the background and the vertices.

Stage 2:

Create the rotation matrices for rotation matrices along X-Y and Z-W axes (w is the 4^{th} dimension). Multiply vertices vector with the rotation matrices to obtain rotated location of the vertices.

Stage 3:

Do stereographic projection for the vertices vector using the formula for stereographic projection rotation.

Stage 4:

Connect the vertices to make it look like a structure. Two for loops are used to connect the inner and the outer cube. Using the third for loop the outer cube and the inner cube is connected.

Stage 5:

Run the tesseract animation

Processing code:

tesseract:

```
float angle = 0;

P4Vector[] points = new P4Vector[16];

void setup() {
  fullScreen(P3D);
  points[0] = new P4Vector(-1, -1, -1, 1);
  points[1] = new P4Vector(1, -1, -1, 1);
  points[2] = new P4Vector(1, 1, -1, 1);
  points[3] = new P4Vector(-1, 1, -1, 1);
  points[4] = new P4Vector(-1, -1, 1, 1);
  points[5] = new P4Vector(1, -1, 1, 1);
  points[6] = new P4Vector(1, 1, 1, 1);
  points[7] = new P4Vector(-1, 1, 1, 1);
  points[8] = new P4Vector(-1, -1, -1, -1);
  points[9] = new P4Vector(1, -1, -1, -1);
  points[10] = new P4Vector(1, 1, -1, -1);
  points[11] = new P4Vector(-1, 1, -1, -1);
  points[12] = new P4Vector(-1, -1, 1, -1);
  points[13] = new P4Vector(1, -1, 1, -1);
  points[14] = new P4Vector(1, 1, 1, -1);
  points[15] = new P4Vector(-1, 1, 1, -1);
  //create the vertices of the hyper cube
}

void draw() {
  background(0);
  translate(width/2, height/2);
  rotateX(-PI/2);
  PVector[] projected3d = new PVector[16];//to store the projected vector

  for (int i = 0; i < points.length; i++) {
    P4Vector v = points[i];

    float[][] rotationXY = {
      {cos(angle), -sin(angle), 0, 0},
      {sin(angle), cos(angle), 0, 0},
      {0, 0, 1, 0},
      {0, 0, 0, 1}
    };//xy rotation matrix

    float[][] rotationZW = {
      {1, 0, 0, 0},
      {0, 1, 0, 0},
      {0, 0, cos(angle), -sin(angle)},
      {0, 0, sin(angle), cos(angle)}
    };//zw rotation matrix


    P4Vector rotated = matmul(rotationXY, v, true);
    rotated = matmul(rotationZW, rotated, true);//applying rotation
```

```
    float distance = 2;
    float w = 1 / (distance - rotated.w);//Stereographic projection formula

    float[][] projection = {
      {w, 0, 0, 0},
      {0, w, 0, 0},
      {0, 0, w, 0}
    };//4d to 3d projection

    PVector projected = matmul(projection, rotated);
    projected.mult(width/8);
    projected3d[i] = projected;

    stroke(255, 200);
    strokeWeight(32);
    noFill();

    point(projected.x, projected.y, projected.z);
  }

  // Connecting
  for (int i = 0; i < 4; i++) {
    connect(0, i, (i+1) % 4, projected3d );
    connect(0, i+4, ((i+1) % 4)+4, projected3d);
    connect(0, i, i+4, projected3d);
  }

  for (int i = 0; i < 4; i++) {
    connect(8, i, (i+1) % 4, projected3d );
    connect(8, i+4, ((i+1) % 4)+4, projected3d);
    connect(8, i, i+4, projected3d);
  }

  for (int i = 0; i < 8; i++) {
    connect(0, i, i + 8, projected3d);
  }

  //angle = map(mouseX, 0, width, 0, TWO_PI);
  angle += 0.02;
}

void connect(int offset, int i, int j, PVector[] points) {
  PVector a = points[i+offset];
  PVector b = points[j+offset];
  strokeWeight(4);
  stroke(255);
  line(a.x, a.y, a.z, b.x, b.y, b.z);
}
```

P4Vector:

```
class P4Vector {
  float  x, y, z, w;
```

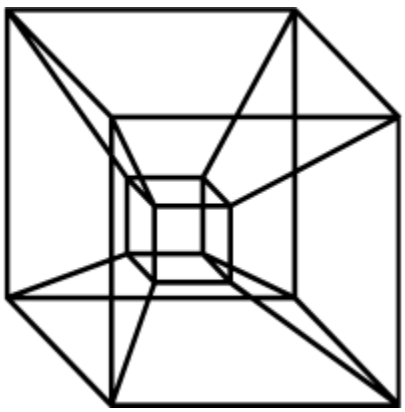
```
  P4Vector(float x, float y, float z, float w) {
    this.x = x;
    this.y = y;
    this.z = z;
    this.w = w;
  }
}//to store the 4d vertices
```

matrix:

```
//to do all matrix computations
float[][] vecToMatrix(P4Vector v) {
  float[][] m = new float[4][1];
  m[0][0] = v.x;
  m[1][0] = v.y;
  m[2][0] = v.z;
  m[3][0] = v.w;
  return m;
}

PVector matrixToVec(float[][] m) {
  PVector v = new PVector();
  v.x = m[0][0];
  v.y = m[1][0];
  v.z = m[2][0];
  return v;
}

P4Vector matrixToVec4(float[][] m) {
  P4Vector v = new P4Vector(0,0,0,0);
  v.x = m[0][0];
  v.y = m[1][0];
  v.z = m[2][0];
  v.w = m[3][0];
  return v;
}


void logMatrix(float[][] m) {
  int cols = m[0].length;
  int rows = m.length;
  println(rows + "x" + cols);
  println("----------------");
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      print(m[i][j] + " ");
    }
    println();
  }
  println();
}
```

```
PVector matmul(float[][] a, P4Vector b) {
  float[][] m = vecToMatrix(b);
  return matrixToVec(matmul(a, m));
}

P4Vector matmul(float[][] a, P4Vector b, boolean fourth) {
  float[][] m = vecToMatrix(b);
  return matrixToVec4(matmul(a, m));
}

float[][] matmul(float[][] a, float[][] b) {
  int colsA = a[0].length;
  int rowsA = a.length;
  int colsB = b[0].length;
  int rowsB = b.length;

  if (colsA != rowsB) {
    println("Columns of A must match rows of B");
    return null;
  }

  float result[][] = new float[rowsA][colsB];

  for (int i = 0; i < rowsA; i++) {
    for (int j = 0; j < colsB; j++) {
      float sum = 0;
      for (int k = 0; k < colsA; k++) {
        sum += a[i][k] * b[k][j];
      }
      result[i][j] = sum;
    }
  }
  return result;
}
```

OUTPUT



PART-2

NCBI Blast Search

BLAST (Basic Local Alignment Search Tool) is an online search tool provided by NCBI that allows you to

"find regions of similarity between biological sequences". It accepts nucleotide as well as protein

sequences. The NCBI maintains a huge database of biological sequences, against which it compares the

query sequences in order to find the most similar ones.

Roll No	9-digit code	Accession No.	Max Score	Total Score	Query Cover	Percent Identity	E value	Organism	'barcode' region
61	QSP006851	NR_043617.1	2700	2700	100%	100%	0	Shewanella irciniae strain	16S ribosomal RNA
61	QMU000612	MF435716.1	1022	1022	100%	100%	0	Parishia sp. JH-2017	rbcL
61	GOH000009	NG_063394.1	3290	3290	100%	100%	0	Candida chilensis	18S ribosomal RNA

Justification:

I had chosen the results (which is shown in the above diagram) by analysing all the parameters in the BLAST search. And I had come to a conclusion that the organism which have maximum value for **max score**, **total score**, **query cover**, **percent Identity** and have minimum **E value** is preferred more to be selected as the perfect match.

And also, I saw that which organism has less gaps between the sequences. All the three sequences which I have listed has 0 gap (This can be observed by seeing the alignment section of BLAST search result page).