Quellcode in Processing

Bubble Script

Beispiel:

```
size(400,300);
 background(255);
}

void draw(){
 for (int i=0; i < 50; i++){
   x = random(10,390);          // random() generiert Zufallswerte
   y = random(10,290);
   d = random(10,75);
   bubbleColor = (int)random(0,255);   // float wird zu int

  Bubble b = new Bubble(d, x, y, color(bubbleColor));
        b.display();
 }

 noLoop();
}
```

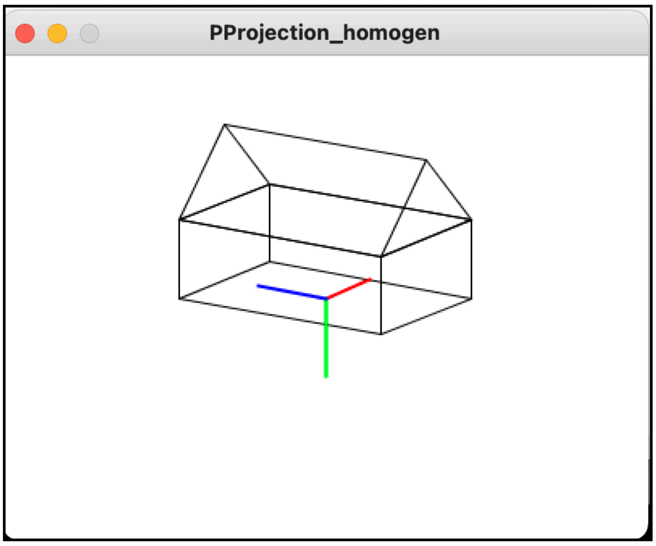
Reiter: BubbleClass

```
class Bubble {
  float x;
  float y;
  float diameter;
  color bubcol;

  Bubble(float tempD, float tempX, float tempY, color tempcol){
  x = tempX;
  y = tempY;
  diameter = tempD;
  bubcol = tempcol;
  }

  void display(){
    stroke(0);
    fill(bubcol);
    ellipse(x,y,diameter,diameter);
  }
}
```

PProjection_homogen

Beispiel:

```
float[] y = {0,1,0};
float[] lookAt = {0,0,0};

float unit = 50;

void setup(){
  size(400,300);
};

void draw(){
   background(255);
   translate(width/2,height/2);

   float[] o = {-1.5,-0.5,-1};
   float[] blickrichtung = lookAtVector(o, lookAt);

   float[] w = normVec(blickrichtung);
   float[] u = vectorProduct(y, w);
         u = normVec(u);
   float[] v = vectorProduct(w,u);

   float[][] TM = worldToCameraMatrix(u,v,w,o);

// Initialisierung der Objekte
   Cube c = new Cube(unit, color(0), 2,1,3, 0, 0,0,0);
   Prism p = new Prism(unit, color(0), 2,1,3, 0, 0,-1,0);

   projection(TM, c);
   projection(TM, p);

   c.drawVertex(c);
   p.drawVertex(p);

   worldKOS(unit, TM, 2);
   noLoop();
}
```



Reiter: Camera

```
float[] vectorProduct(float[] a , float[] b){

  int n = a.length;
  int m = b.length;

  if(n != 3 | m !=3){
    println("vector length must be 3" );
    return null;
  };

  float[] result = new float[3];

  result[0] = a[1]*b[2] - a[2]*b[1];
  result[1] = a[2]*b[0] - a[0]*b[2];
  result[2] = a[0]*b[1] - a[1]*b[0];
  return result;
}


float[] normVec(float[] v){
  int n = v.length;

  float sum=0;

  for(int i=0; i<n;i++){
    sum = v[i]*v[i] + sum;
  }

  float vLength = sqrt(sum);

  float[] result = new float[n];

  for(int i=0; i<n;i++){
    result[i] = v[i]/vLength;
  }

  return result;
}


float[][] worldToCameraMatrix(float[] u, float[] v, float[] w, float[] CameraPosition){

  float[][] rotMatrix = {{u[0],u[1],u[2],0},
                         {v[0],v[1],v[2],0},
                         {w[0],w[1],w[2],0},
                         {  0,   0,    0,1}};

  float[][] transMatrix = {{1,0,0,-CameraPosition[0]},
                           {0,1,0,-CameraPosition[1]},
                           {0,0,1,-CameraPosition[2]},
                           {0,0,0,   1}};

  float[][] result = matmul(rotMatrix, transMatrix) ;

  return result;
}
```

```
float[] lookAtVector(float[] CameraPosition, float[] lookPoint){

  int n = CameraPosition.length;
  float[] blickrichtung = new float[n];

  for(int i=0; i<n; i++){
    blickrichtung[i] = lookPoint[i] - CameraPosition[i];
  }

  return blickrichtung;
}
```

Reiter: Matrics

```
//Matrixmultiplikation
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("Spalten von A müssen mit den Zeilen von B übereinstimmen");
    return null;
  }


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

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


void printMatrix(float[][] a){
  int colsA = a[0].length;
  int rowsA = a.length;

  for(int i=0; i < rowsA; i++){
    for(int j = 0; j < colsA; j++){
      print(a[i][j] + " ");
    }
    println(" ");
  }
}
```

```
float[] MatDotPoint(float[][] mat, float[] point){
  int k = point.length;
  int n = mat.length;
  int m = mat[0].length;

  if(k != m){
   println("Anzahl der Spalten muss mit der Anzahl der Koordinaten übereinstimmen");
   return null;
  }

  float[] result = new float[n];

  for(int i=0; i < n; i++){
    float sum = 0;
    for(int j=0; j < m; j++){
      sum += mat[i][j] * point[j];
    }
    result[i] = sum;
  }

  return result;
}


float[][] productMatrix(ArrayList<float[][]> listOfMatrix){

  float[][] res = matmul(listOfMatrix.get(1), listOfMatrix.get(0));

  int n = listOfMatrix.size();

  for(int i=2; i < n; i++){
    res = matmul(listOfMatrix.get(i), res);
  }

  return res;
}


// Entweder eine beliebige Matrix oder die WorldToCam Matrix
void projection(float[][] mat, Shapes shape){
  float[][] pointlist = shape.points;
  int n = pointlist.length;

  for(int i=0; i < n; i++){
    pointlist[i] = MatDotPoint(mat, pointlist[i]);
  }
}
```

Reiter: Shape

```
class Shapes{
  //Einheit;
  float e;
  color linecolor;
  float[][] points;
  int[][] edges;

  //Transformation
  float sca, scb, scc; // geht nicht als float[], da Array im Constructor nicht erzeugt wird
  float angle;
  float vx, vy, vz;

  Shapes(float unit, color linecol,
        float sc_a, float sc_b, float sc_c, float ang,
            float v_x, float v_y, float v_z){
    e = unit;
    linecolor = linecol;
    sca = sc_a;
    scb = sc_b;
    scc = sc_c;
    angle = ang;
    vx = v_x;
    vy = v_y;
    vz = v_z;
  }
  void drawLine(float[] p, float[] q, color linecolor){
    stroke(linecolor);

    // Nur die x,y Koordinaten werden gezeichnet
      line(p[0], p[1], q[0], q[1]);

  }


 void drawVertex(Shapes shape){
    points = shape.points;
    edges = shape.edges;
    linecolor = shape.linecolor;

   for(int i=0; i < edges.length; i++){
   drawLine(points[edges[i][0]], points[edges[i][1]], linecolor);
    }
  }


void transform(float sca, float scb, float scc,
    float angle, float vx, float vy, float vz, float[][] points, float e){

    // Erstelle homogene Transformationsmatrizen
    float[][] M_scale = MScale(sca, scb, scc);
    float[][] M_rot_y = MRotate(angle, 'y');
    float[][] M_shift = MShift(vx*e, vy*e, vz*e);

    ArrayList<float[][]> matrixlist = new ArrayList<float[][]>(java.util.Arrays.asList(
                                     M_scale,
                                     M_rot_y,
                                     M_shift
                                            ));
```

```
// Erstelle Transformationsmatrix
    float[][] Matrix = productMatrix(matrixlist) ;

    // Multipiliziere jeden Eckpunkt mit der Transformationsmatrix
    for(int i=0; i < points.length; i++){
      points[i] = MatDotPoint(Matrix,points[i]);
    }
  }

}


class Cube extends Shapes{

  Cube(float e, color linecolor,
       float sca, float scb, float scc, float angle,
       float vx, float vy, float vz){
    super(e, linecolor, sca, scb, scc, angle, vx, vy, vz);

    // homogene Koordinaten
    float[][] tempPoints = {{0.5*e, 0,0.5*e,1}, {-0.5*e, 0, 0.5*e, 1}, {-0.5*e, 0, -0.5*e, 1},  {0.5*e, 0,  -0.5*e, 1},
                           {0.5*e,-e,0.5*e, 1}, {-0.5*e, 0-e, 0.5*e, 1}, {-0.5*e, -e, -0.5*e, 1}, {0.5*e, -e, -0.5*e, 1}};
    points = tempPoints;

    int[][] tempEdges = {{0,1}, {1,2}, {2,3}, {3,0},
            {4,5}, {5,6}, {6,7}, {7,4},
            {0,4}, {1,5}, {2,6}, {3,7} };
    edges = tempEdges;

    transform(sca, scb, scc, angle, vx, vy, vz, points, e);
  }
}


class Pyramid extends Shapes{

  Pyramid(float e, color linecolor,
       float sca, float scb, float scc, float angle,
       float vx, float vy, float vz){
    super(e, linecolor, sca, scb, scc, angle, vx, vy, vz);

    float[][] tempPoints = {{0.5*e, 0, 0.5*e, 1}, {-0.5*e, 0, 0.5*e, 1}, {-0.5*e, 0, -0.5*e, 1}, {0.5*e, 0, -0.5*e, 1},
                      {0, -e, 0, 1}};
          points = tempPoints;
    int[][] tempEdges = {{0,1}, {1,2}, {2,3}, {3,0},
                   {0,4}, {1,4}, {2,4}, {3,4}};
          edges = tempEdges;

    transform(sca, scb, scc, angle, vx, vy, vz, points, e);
  }
}


class Prism extends Shapes{

  Prism(float e, color linecolor,
       float sca, float scb, float scc, float angle,
       float vx, float vy, float vz){
    super(e, linecolor, sca, scb, scc, angle, vx, vy, vz);
    float[][] tempPoints = {{0.5*e, 0, 0.5*e, 1}, {-0.5*e, 0, 0.5*e, 1}, {-0.5*e, 0, -0.5*e, 1}, {0.5*e, 0, -0.5*e, 1},
                          {0, -e, -0.5*e, 1}, {0, -e, 0.5*e, 1}};
              points = tempPoints;
    int[][] tempEdges = {{0,1}, {1,2}, {2,3}, {3,0},
                        {0,5}, {1,5}, {2,4}, {3,4},
                        {4,5}};
            edges = tempEdges;

    transform(sca, scb, scc, angle, vx, vy, vz, points, e);
  }
}


void worldKOS(float e, float[][] worldToCamMatrix, float linewidth){

      float[] origin = {0, 0, 0, 1};
      float[] e_x = {e, 0, 0, 1};
      float[] e_y = {0, e, 0, 1};
      float[] e_z = {0, 0, e, 1};

      // Parallelprojektion der Einheitsvektoren
      origin = MatDotPoint(worldToCamMatrix, origin);
      e_x = MatDotPoint(worldToCamMatrix, e_x);
      e_y = MatDotPoint(worldToCamMatrix, e_y);
      e_z = MatDotPoint(worldToCamMatrix, e_z);

      // Zeichnen des KOS in den entsprechenden Farben
      strokeWeight(linewidth);
      stroke(255,0,0);
      line(origin[0], origin[1], e_x[0], e_x[1]);

      stroke(0,255,0);
      line(origin[0], origin[1], e_y[0], e_y[1]);

      stroke(0,0,255);
      line(origin[0], origin[1], e_z[0], e_z[1]);

      strokeWeight(1); // Linienbreite zurueck auf die Standardeinstellung
  }
```

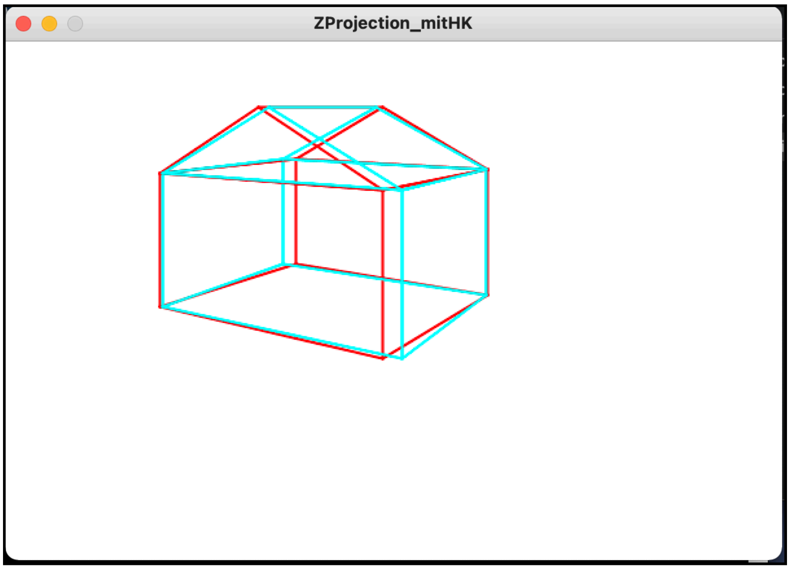
Reiter: Transformation

```
//Transformationsmatrizen

 float[][] MScale(float sca, float scb, float scc){

   float[][] M = {{sca, 0, 0, 0,},
                  {0, scb, 0, 0},
                  {0, 0, scc, 0},
                  {0, 0, 0, 1}};
   return M;
 }

 float[][] MRotate(float angle, char axis){

   float alpha = PI * angle/180;   //Deg to Rad

   if(axis == 'x'){
     float[][] M = {{1, 0, 0, 0,},
                    {0, cos(alpha), -sin(alpha), 0},
                    {0, sin(alpha), cos(alpha), 0},
                    {0, 0, 0, 1}};

                  return M;
   }

   if(axis == 'y'){
    float[][] M = {{cos(alpha), 0, sin(alpha), 0},
                   {0, 1, 0, 0},
                   {-sin(alpha), 0, cos(alpha), 0},
                   {0, 0, 0, 1}};

                  return M;
   }

   if(axis == 'z'){
    float[][] M = {{cos(alpha), -sin(alpha), 0, 0},
                   {sin(alpha), cos(alpha), 0, 0},
                   {0, 0, 1, 0},
                   {0, 0, 0, 1}};

                  return M;
   }

   else{
     println("wrong name of axis! Choose x, y or z");
     return null;
   }
 }

 float[][] MShift(float vx, float vy, float vz){

   float[][] M = {{1, 0, 0, vx},
                  {0, 1, 0, vy},
                  {0, 0, 1, vz},
                  {0, 0, 0, 1}};
   return M;
 }
```

ZProjection_mitHK

Beispiel:

```
float unit = 50.0;

int[] red = {255,0,0,255};
int[] cyan = {0,255,255,255};

float[] Z1 = {-0.5*unit,-3*unit,-10*unit};   // Cyan
float[] Z2 = {0.5*unit,-3*unit,-10*unit}; // Rot

void setup(){
   size(600,400);
};

void draw(){
   background(255);
   translate(width/2,height/2);

// Initialisierung der Objekte
// Cube(Einheit, Farbe, Breite, Hoehe, Tiefe, Drehwinkel in Grad, vx, vy, vz, ax)

   Cube Haus1_c = new Cube(unit, cyan, 4, 2, 3, 30, -1, 0, 0, 0);
   Cube Haus1_r = new Cube(unit, red, 4, 2, 3, 30, -1, 0, 0, 0);

   Prism Dach1_c = new Prism(unit, cyan, 4, 1, 3, 30, -1, -2, 0, 0);
   Prism Dach1_r = new Prism(unit, red, 4, 1, 3, 30, -1, -2, 0, 0);

// Zentralprojektion auf die xy Ebene

 strokeWeight(2);
  CentralProjection(Z1, Haus1_c) ;
  CentralProjection(Z1, Dach1_c) ;

  CentralProjection(Z2, Haus1_r) ;
  CentralProjection(Z2, Dach1_r) ;

  Haus1_r.drawVertex() ;
  Haus1_c.drawVertex() ;
  Dach1_r.drawVertex() ;
  Dach1_c.drawVertex() ;

 noLoop();
}
```



Reiter Matrics wie zuvor (ohne letzte Methode *Projection*), S. 5

Reiter Shape wie zuvor, S. 7

Reiter Transformation wie zuvor, S. 10, mit folgender, zusätzlicher Methode:

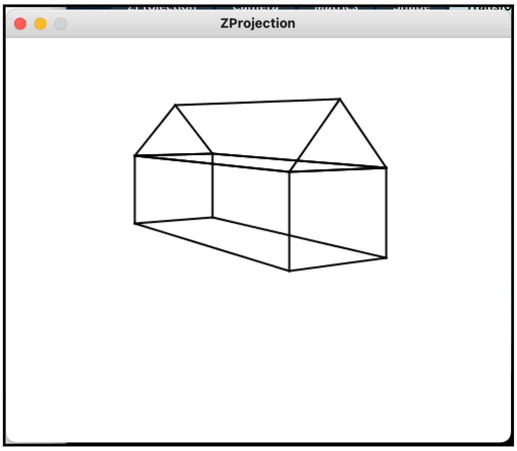
```
// Zentralprojektion auf xy Ebene mit homogene Koordinaten, aber ohne Transformationsmatrix

void CentralProjection(float[] zentrum, Shapes shape){
   float[][] pointlist = shape.points;
   int n = pointlist.length;

   float l_1 = zentrum[0];
   float l_2 = zentrum[1];
   float l_3 = zentrum[2];

  for(int i=0; i < n; i++){

   float w_1 = pointlist[i][0];
   float w_2 = pointlist[i][1];
   float w_3 = pointlist[i][2];

   float vorfaktor = 1 / (l_3- w_3);

   pointlist[i][0] = vorfaktor * (l_3 * w_1 - l_1 * w_3);
   pointlist[i][1] = vorfaktor * (l_3 * w_2 - l_2 * w_3);
   pointlist[i][2] = 0;
  };
}
```

ZProjection

Zentralprojektion mit homogenen Koordinaten und Kamerakoordinatensystem

Beispiel:

```
float unit = 40.0;
float[] y = {0,1,0};
float[] lookAt = {0,-3,0};

void setup(){
   size(500,400);
};

void draw(){
  background(255);
  translate(width/2,height/2);

  float[] Z = {-10, -3, -8};

   float[] o = Z;
   float[] blickrichtung = lookAtVector(o, lookAt);

  float[] w = normVec(blickrichtung);
  float[] u = vectorProduct(y, w);
          u = normVec(u);
  float[] v = vectorProduct(w,u);

  float[][] TM = worldToCameraMatrix(u,v,w,o);

// Initialisierung der Objekte

    Cube c = new Cube(unit, color(0,0,0), 3,2,3, 40, 0,0,0);
    Prism p = new Prism(unit, color(0,0,0), 3,1.5,3, 40, 0,-2,0);

    projection(TM, c);
    projection(TM, p);

// Zentralprojektion auf die xy Ebene

    CentralProjection(Z, c);
    CentralProjection(Z, p);

    strokeWeight(2);

    c.drawVertex(c);
    p.drawVertex(p);

    noLoop();
}
```



Reiter: Matrics wie zuvor, S. 5
Reiter: Camera wie zuvor, S. 4
Reiter: Shape wie zuvor, S. 7
Reiter: Transformation wie zuvor, S. 10, mit zusätzlicher, folgender Methode:

```
// Matrix für Zentralprojektion mit homogenen Koordinaten
void CentralProjection(float[] zentrum, Shapes shape){

  float unit = shape.e;

  float z1 = zentrum[0] * unit ;
  float z2 = zentrum[1] * unit ;
  float z3 = zentrum[2] * unit ;

  float[][] M = {{z3, 0, -z1, 0},
                {0, z3, -z2, 0},
                {0, 0, 0, 0},
                {0, 0, -1, z3}};

  float[][] pointlist = shape.points;
  int n = pointlist.length;

  for(int i=0; i < n; i++){

    pointlist[i] = MatDotPoint(M, pointlist[i]);

    float t = pointlist[i][3];

    pointlist[i][0] = pointlist[i][0] /t  ;
    pointlist[i][1] = pointlist[i][1] /t  ;
    pointlist[i][2] = pointlist[i][2] /t  ;
    pointlist[i][3] = 1 ;
  };
}
```