cobaea scandens | konzept

diplomwahlfacharbeit processing | programmieren statt zeichnen

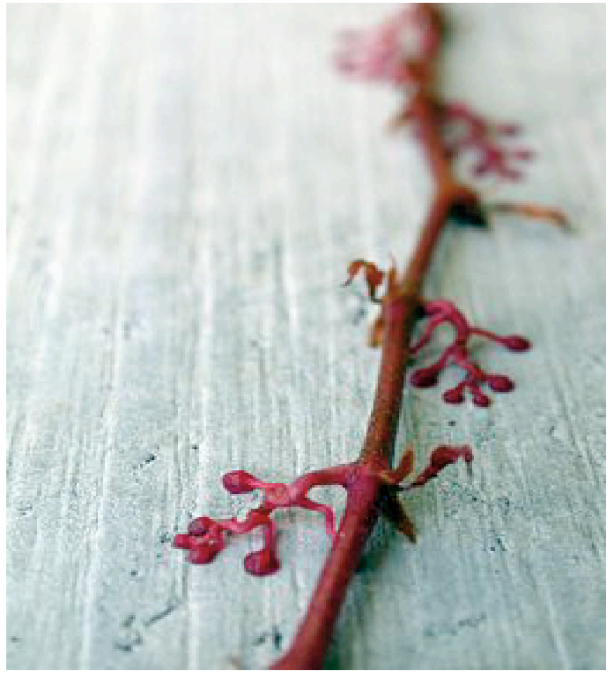
professur für caad | prof. ludger hovestadt
stud. kaj blattner | dina hool | renate walter

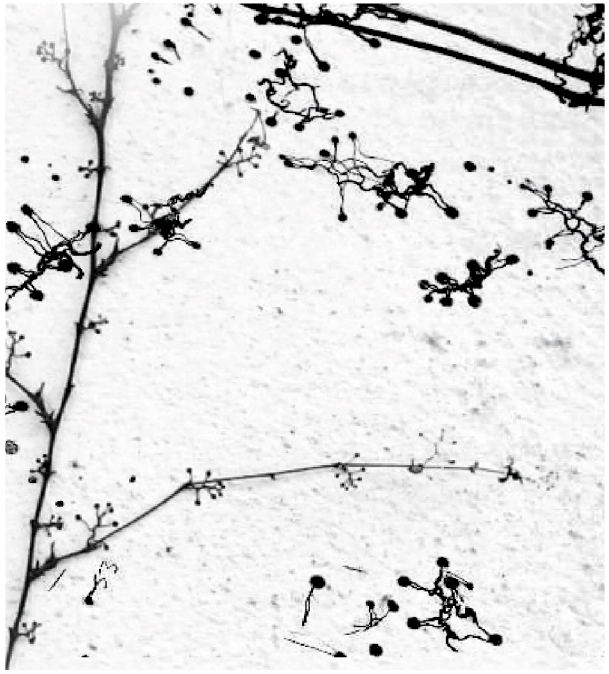
ss 07

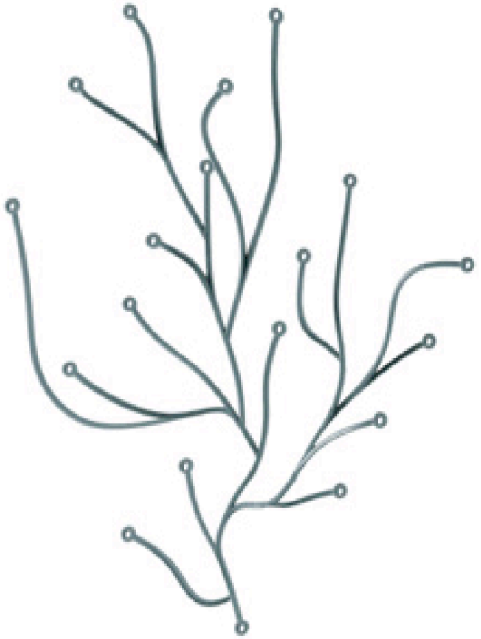
Als Ausgangslage dient uns das Konzept des kletterpflanzenartigen Gefüges. Die Idee ist es nun ein Programm zu schreiben, das als Ausgabe eine gewachsene Struktur hat. Wobei gewisse Parameter bestimmt werden, zum Beispiel das Teilungsverhältnis der Äste, der Winkel bei dem ein neuer an einen alten Ast trifft, um wie viel die Länge des nächsten Ast zum vorhergehenden abnehmen soll usw. Die einzelnen Äste werden nur bis zu einer vorher bestimmten Pixellänge gezeichnet. Und erhalten dann einen Abschluss. Andere Einflussgrössen können ganz oder in einem definierten Bereich dem Zufall überlassen werden.
Wir erhalten Strukturen, die gewisse Regelmässigkeiten enthalten in ihrer ausgegebenen Form aber jedes Mal anders sind.

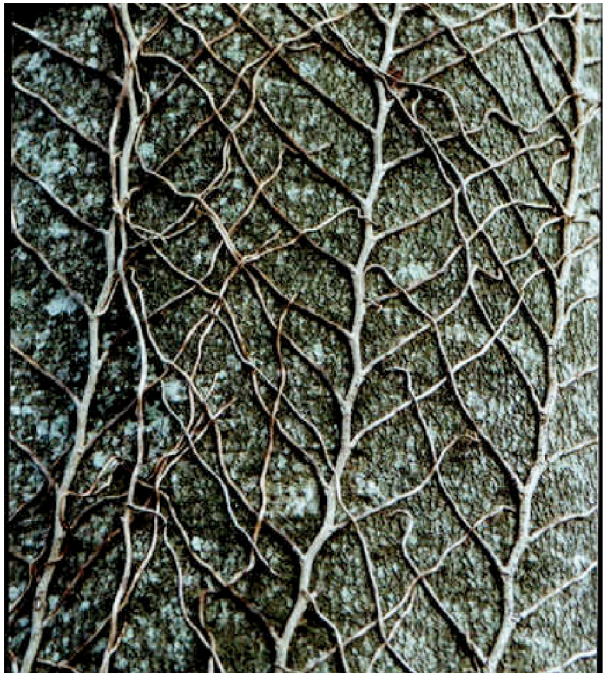
Unterschiedliche Verwendungszwecke sind vorstellbar/denkbar/möglich;

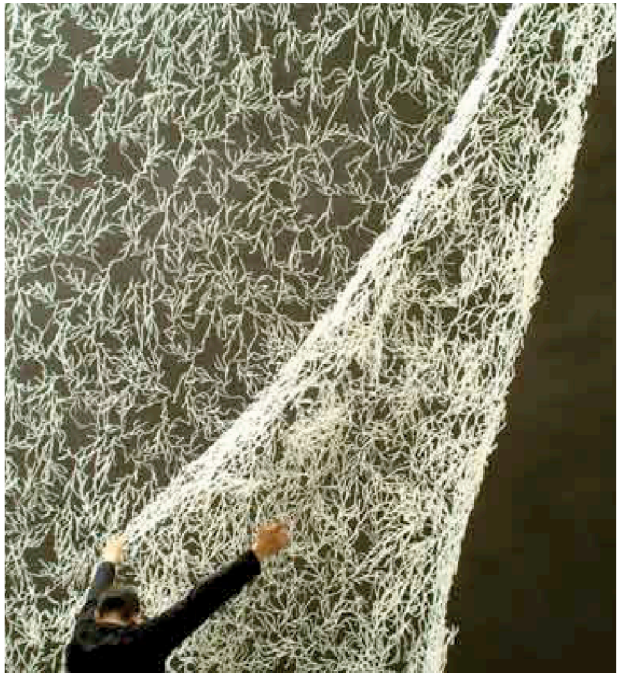
Raumteiler, Vorhang:	Die Ausgabe wird als Vektorzeichnung Gesichert und mit dem 2D-Schneidplotter bearbeitet und gefaltet, es entsteht eine räumliche Struktur
Tapete, Muster:	Auf eine Klebefolie gedruckt, kann das Muster appliziert werden. Durch die importierte Vorlage können Öffnungen ausgespart werden.

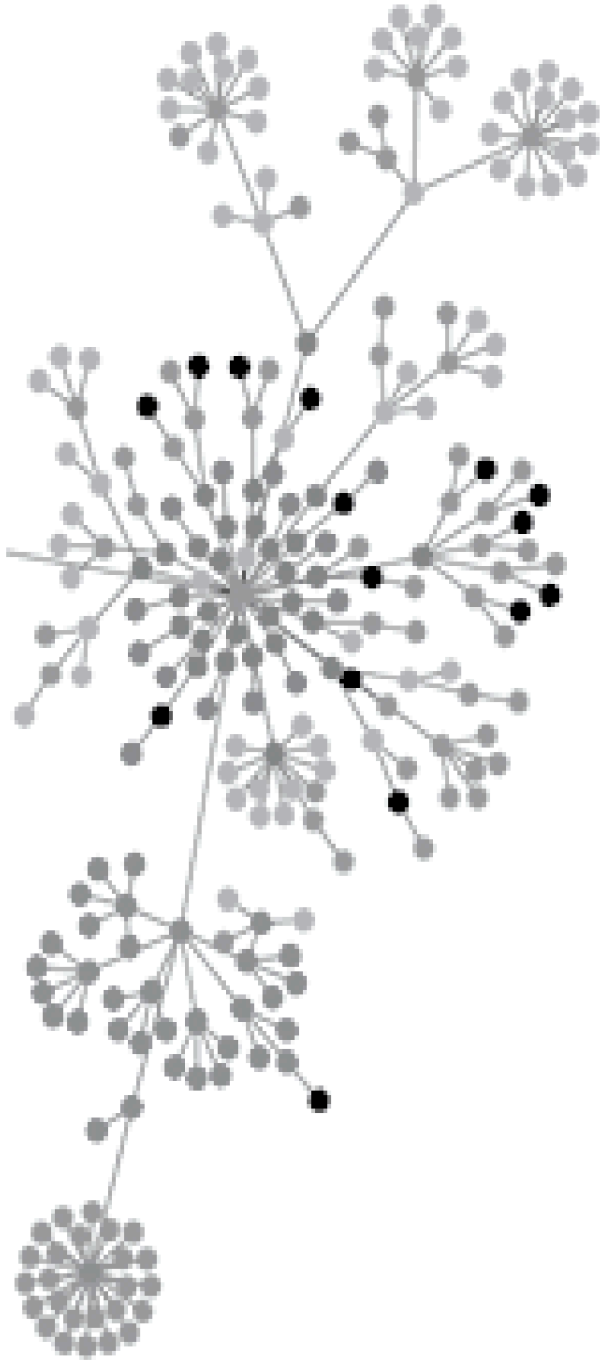


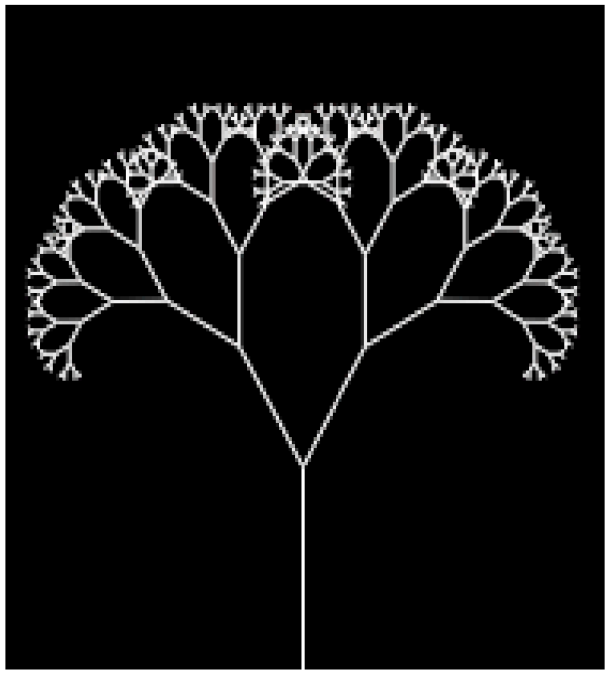


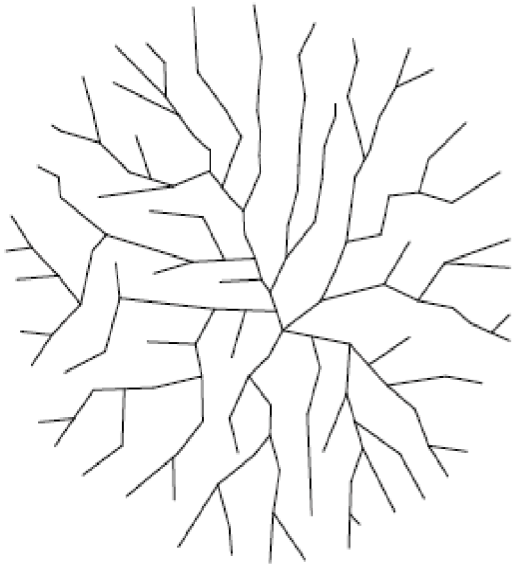


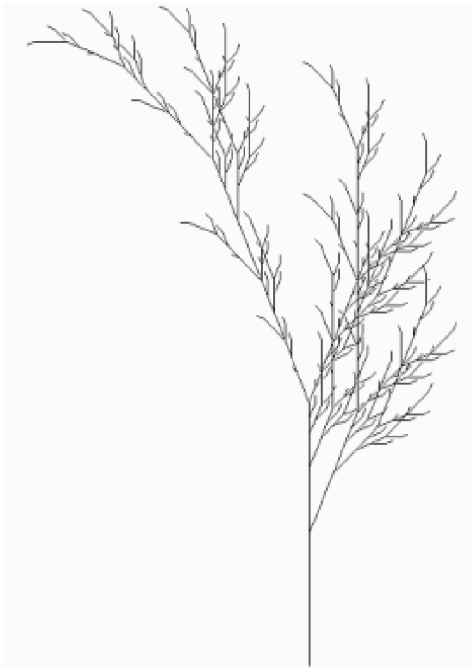


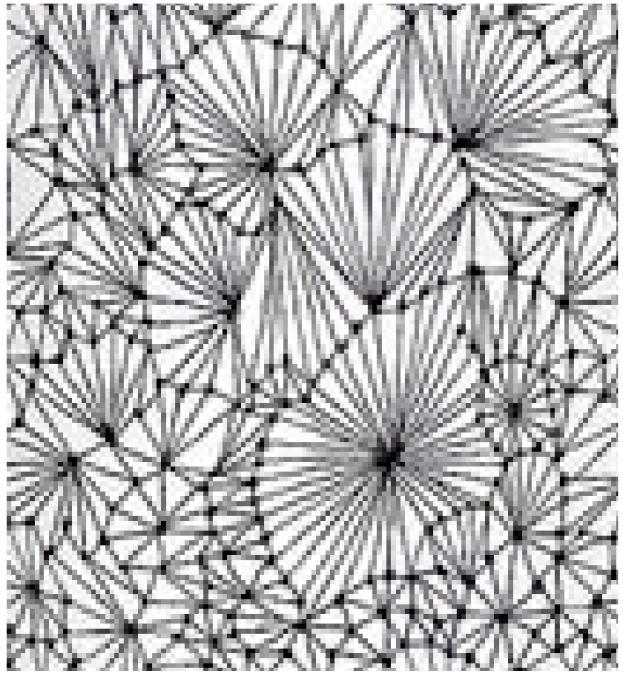






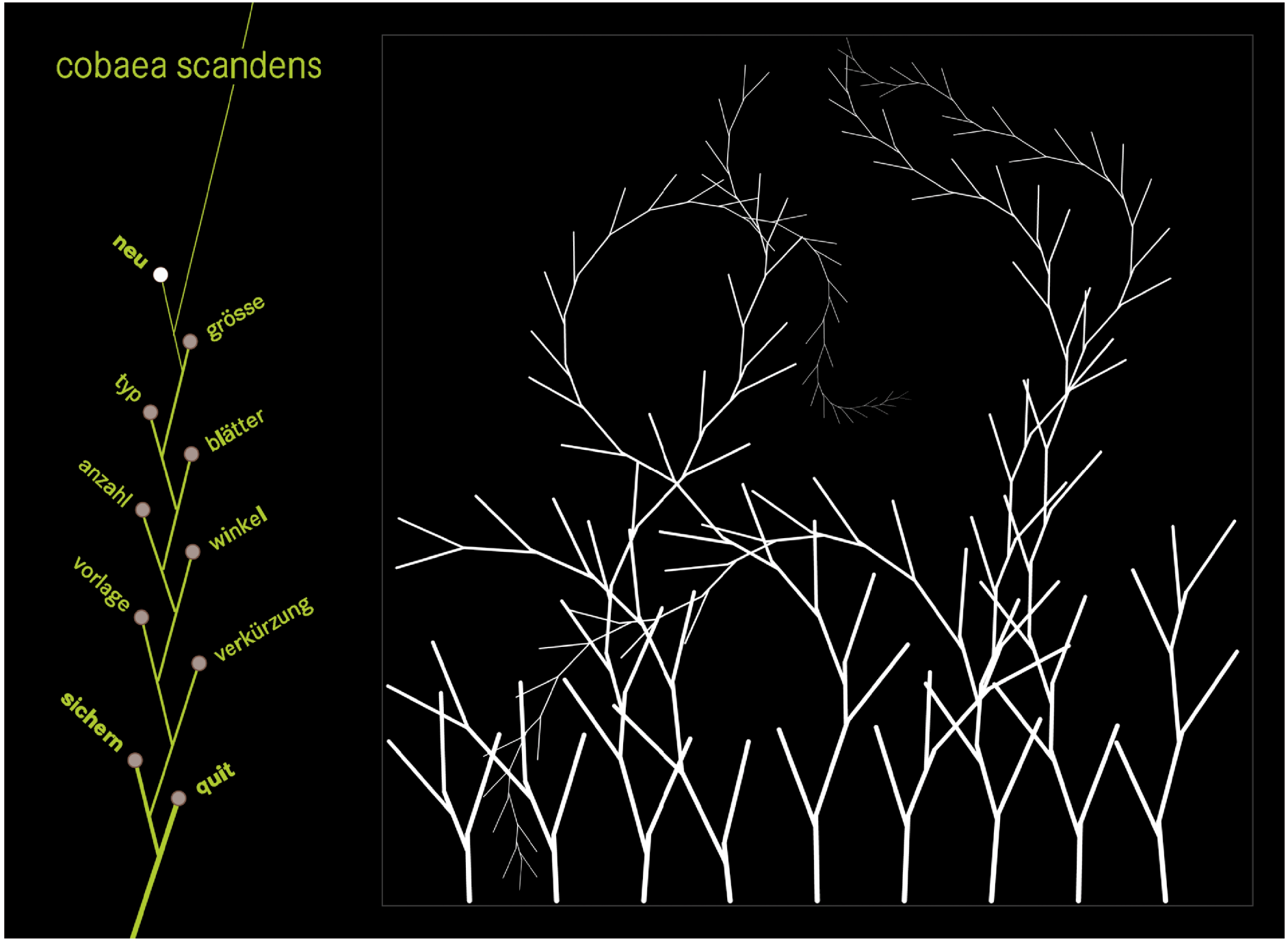






Das Programmfenster ist in einen Steuerungsteil und ein Ausgabefenster gegliedert. Im Steuerungsteil können vom Baumtyp über die Anzahl der Bäume verschiedenste Parameter wie die Grösse, die Blätter, die Winkel der Gabelungen und die Verkürzung der Äste verändert werden. Unter „sichern“ wird ein pdf-File erzeugt das für die Ausgabe über einen Drucker oder Schneideplotter weiter verarbeitet werden kann.

Struktur
Das Programm ist neben dem Hauptprogrammteil auf drei Klassen, Branch, Edge und Point aufgebaut.
Die drei Klassen sind hierarchisch miteinander verbunden, das heisst in Edge wird mit Points operiert und in Branch mit Edges. Im Hauptprogrammteil sind alle zum generieren der Bäume notwendigen Variablen enthalten und können über das Menu verändert werden. Bevor die Bäume gezeichnet werden, wird eine Liste mit allen auszugebenden Bäumen gemacht. Der Bauplan eines jeden Baumes wird in der Klasse Branch in Listen geschrieben. Jede Branch enthält vier Listen. Die printList enthält den kompletten Bauplan und wird folglich immer gezeichnet wird. Die endingList und pointList enthalten die Information der Blätter und können zusätzlich gezeichnet werden. Die constructList enthält nur diejenigen Kannten die noch nicht geteilt wurden und ist nur Hilfsmittel um das Wachstum bzw. den Teilungsvorgang zu steuern. Eine Methode von Branch entscheidet aufgrund dessen Typs, welche Kannten wievielmal und mit welcher Methode geteilt werden.
Jeder Baum wird mit dem Stamm (edge1) gestartet. Die zu teilenden Kannten werden mit einer Methode in Edge einmal oder zweimal geteilt.
Typ 1 ist eine Grasartige Struktur bei der jede kannte nur einmal geteilt wird. Typ 2 benutzt eine Methode für zweimaliges Teilen. Sie teilt jedoch nur eine der beiden neu entstehenden Kannten weiter. Typ 3 teilt zwei der neu entstandenen Kannten. Es entsteht ein Baumtyp der nicht allzu dicht ist und in der Form stark variiert. Typ 4 teilt zunehmend mehr der entstandenen Kannten und bekommt daher ein dichteres Volumen. Typ 5 teilt alle neu entstanden Kannten zweimal. Es entsteht ein relativ regelmässiger Baumtyp.


cobaea scandens
neu
grösse
typ
blätter
anzahl
winkel
vorlage
verkürzung
sichern
quit

```
//————————————Deklarationen————————

//import processing.pdf.*;
ArrayList numberOfBranch = new ArrayList();



int fensterX = 1000;
int fensterY = 715;
int ausgabeX = 680;
int ausgabeY = 665;

int abstand = (fensterY - ausgabeY)/2;
int ausgabeXpos = fensterX - (ausgabeX + abstand);



//————————————Variablen zur Steuerung————————

float astDicke = 2;                                  //  Dicke beim ersten Stamm
int anzahlBaeume = 1;                                //  Anzahl Bäume
int endPunktVariante = 0;                            //  Astende:    1 = zwei kleine Zweige    2 = Kreise
int branchType = 1;                                  //  Typ 1 = Standard, 2 = einfache Verzweigung, 3 = 2fach Rekursive Verzweigung
int anzahlPunkte = 25;                               //  Verzweigungsgrad
float d = 30;                                        //  Stammlänge
float verkuerzung = 0.94;
float schwindFaktor = 0.9;
float angle = PI/8;



//————————————Abhängige Variablen————————

float punktRadius = astDicke+astDicke;               //  Radius der Kreise (Class Branch)
int distance = ausgabeX / (anzahlBaeume+1);          //  Distanz der Bäume



void setup()
{
  smooth();
  size(fensterX,fensterY);
  //  beginRecord(PDF, „everything.pdf“);            // pdf file
  background(0);
  drawLayoutFix();                                   //  Menu wird gezeichnet    aus dem Tab _LAYOUT
}
```

```
int j =0;
float longestList = 1;

void draw()
{
 drawLayoutChangeable();                                                    // Menubuttons werden gezeichnet   aus dem Tab _LAYOUT
 strokeWeight(1);
 noFill();
 stroke(50);
 rect(ausgabeXpos, abstand,ausgabeX,ausgabeY);
 stroke(255);



 //————————————GENERIEREN DER BÄUME————————————————

 if (longestList > j)
 {

  for (int c = 0; c < anzahlBaeume; c++)
  {
   Branch newBranch = new Branch(anzahlPunkte,branchType);                  // Baum wird definiert
   Point punkt0 = new Point(ausgabeXpos + (c+1)*distance,height-abstand-astDicke);   // Position der ersten 2 Punkte abhängig von Anzahlbäume
   Point punkt1 = new Point(ausgabeXpos + (c+1)*distance+random(-d/10,d/10), height-abstand-random(9*d/10,d));   // Sind gleichmässig an unterer Seite des Ausgabefeldes verteilt
   Edge edge1 = new Edge(punkt0,punkt1,astDicke,newBranch);                 // Erste Kannte (Baumstamm) wird definiert
   newBranch.makeOtherEdges(edge1);                 // Methode, die das Erstellen der Bäume steuert wird aufgerufen. Hauptteil des Baum-generierens wird über andere assen gemacht
    numberOfBranch.add(newBranch);                                          // Kompletter Baum wird in Baumliste geschrieben
  }


  println(j);

  //————————————AUSGABE BÄUME————————————————

  for (int b = 0; b < anzahlBaeume; b++)                                    // Von allen Bäumen wird jeweils eine Kannte gezeichnet
  {
   Branch currentBranch = (Branch)numberOfBranch.get(b);                    // Bäume werden aus der numberOfBrunch (Liste aller Bäume) gelesen
   int c = currentBranch.printList.size();
   if (c > j ){                                                             // Eine Kannte wird aus dem Baum gelesen und gezeichnet
    Edge testEdge = (Edge)currentBranch.printList.get(j);
    testEdge.drawMe(255,255,255);
    if (endPunktVariante > 0)                                               // Blattyp wird gezeichnet falls endPunktVariante 1 oder 2
    {
     if (endPunktVariante == 1)
     {
      if (currentBranch.endingList.size() > j+1)
      {
       testEdge = (Edge)currentBranch.endingList.get(j);
```

```
            testEdge.drawMe(255,255,255);                    // Kleine Zweige werden gezeichnet
           }
          }
          if (endPunktVariante == 2)
          {
           if (j < currentBranch.endingList.size()-1)
           {
            testEdge = (Edge)currentBranch.endingList.get(j);
            testEdge.drawMe(punktRadius,punktRadius,255,255,255, j);   // Methode die Ellipsen zeichnet wird aufgerufen
           }
          }
         }
        }


        //saveFrame();

        Branch firstBranch = (Branch)numberOfBranch.get(0);   // ist zum Stoppen der prozedur wenn alles gezeichnet ist
        longestList= firstBranch.printList.size();            // herausfinden, welcher baum die meisten Kannten hat

        for ( int d = 0; d < anzahlBaeume; d++)
        {
         firstBranch = (Branch)numberOfBranch.get(d);
         if (firstBranch.printList.size() > longestList)
         {
          longestList = firstBranch.printList.size();
         }
        }
       }

       println(longestList);

       if (j < longestList)
       {
        j++;
       }
       else {
        noLoop();
       }
      }
    }


    void mousePressed()
    {
     if (mouseButton == LEFT)
     {
      if (mouseX > 85 && mouseX < 130)                    // NEU
      {
       if (mouseY > 175 && mouseY < 220)
```

```
  {
    newAutput();
  }
}


if (mouseX > 85 && mouseX < 125)                                    // TYP
{
  if (mouseY > 285 && mouseY < 325)
  {
    if (branchType < 5)
    {
      branchType = branchType + 1;
    }
    else {
      branchType = 1;
      anzahlPunkte = 25;
    }
    if (branchType == 4 || branchType == 5)
    {
      anzahlPunkte = 7;
    }
    newAutput();
  }
}


if (mouseX > 55 && mouseX < 115)                                    // ANZAHL
{
  if (mouseY > 350 && mouseY < 400)
  {
    if (anzahlBaeume < 12)
    {
      anzahlBaeume = anzahlBaeume +1;
    }
    else {
      anzahlBaeume = 1;
    }
    distance = ausgabeX / (anzahlBaeume+1);
    newAutput();
  }
}


if (mouseX > 55 && mouseX < 120)                                    // VORLAGE
{
  if (mouseY > 425 && mouseY < 480)
  {

    displayImage();
```

```
  }
}

if (mouseX > 140 && mouseX < 205)                    // GRÖSSE
{
  if (mouseY > 220 && mouseY < 270)
  {
    if (d < 130)
    {
      d = d+20;
      astDicke = astDicke + 3/2;
      punktRadius = astDicke+astDicke;
    }
    else {
      d = 30;
      astDicke = 2;
      punktRadius = astDicke+astDicke;
    }
    newAutput();
  }
}

if (mouseX > 138 && mouseX < 205)                    // BLÄTTER
{
  if (mouseY > 310 && mouseY < 355)
  {

    if (endPunktVariante < 2)
    {
      endPunktVariante = endPunktVariante +1;
    }
    else {
      endPunktVariante = 0;
    }
    punktRadius = astDicke+astDicke;
    newAutput();
  }
}

if (mouseX > 138 && mouseX < 205)                    // WINKEL
{
  if (mouseY > 385 && mouseY < 430)
  {
    if (angle > PI/18)
    {
      angle = angle - PI/18;
    }
```

```
      else {
        angle = PI/4;
      }
      newAutput();
    }
  }


  if (mouseX > 144 && mouseX < 245)                    // VERKÜRZUNG
  {
    if (mouseY > 455 && mouseY < 515)
    {
      if (verkuerzung > 0.87)
      {
        verkuerzung = verkuerzung - 0.03;
      }
      else {
        verkuerzung = 1;
      }
      newAutput();
    }
  }


  if (mouseX > 45 && mouseX < 110)                     // SICHERN
  {
    if (mouseY > 520 && mouseY < 590)
    {
      //  endRecord();
    }
  }


  if (mouseX > 130 && mouseX < 182)                    // QUIT
  {
    if (mouseY > 575 && mouseY < 620)
    {
      exit();
    }
  }
 }

}


//————————Funktion, welche erneut neue Bäume generiert

void newAutput()
{
 j =0;
```

```
  fill(0);
  noStroke();
  rect(ausgabeXpos -5, abstand -5,ausgabeX +10,ausgabeY +10);
  noFill();
  stroke(50);
  strokeWeight(1);
  longestList = 1;
  punktRadius = astDicke+astDicke;
  rect(ausgabeXpos, abstand,ausgabeX,ausgabeY);
  numberOfBranch.clear();
}

void displayImage()
{
  j =0;
  PImage bild;
  bild = loadImage(„test.jpg“);
  float hoehe = bild.height;
  float breite = bild.width;
  println(hoehe);
  println(breite);
  float scaleX = ausgabeX/breite;
  println(scaleX);
  image(bild, ausgabeXpos, fensterY-abstand-hoehe*scaleX, breite*scaleX, hoehe*scaleX);
  noFill();
  stroke(50);
  strokeWeight(1);
  longestList = 1;
  punktRadius = astDicke+astDicke;
  rect(ausgabeXpos, abstand,ausgabeX,ausgabeY);
  numberOfBranch.clear();
}




class Branch
{
  ArrayList constructList = new ArrayList();                    // Liste Enthält Kanten, die noch geteilt werden dürfen
  ArrayList pointList = new ArrayList();                        // Liste Enthält alle Punkte bei den Gabelungen
  ArrayList printList = new ArrayList();                        // Liste enthält alle Kanten des Baumes: Entspricht dem Bauplan um Baum zu zeichnen
  ArrayList endingList = new ArrayList();                       // Liste enthält alle Kanten die für jeweiligen Blatttyp gebraucht wird
  int type;
  int branchLength;


  Branch(int numberEdge, int branchType)
  {
    branchLength = numberEdge;
```

```
  type = branchType;
}


//———————Erstellen der Branch Listen

void makeOtherEdges(Edge startEdge)                    // Listen aller Kanten eines Baumes werden nach aufgeruftem Branchtyp erstellt.
{
  Edge edge1 = startEdge;
  constructList.add(edge1);                            //  Liste Enthält Kanten, die noch geteilt werden dürfen
  printList.add(edge1);                                //  Liste enthält alle Kanten des Baumes: Entspricht dem Bauplan um Baum zu zeichnen


  if (type ==1)                                        //  Type 1 ruft Methode für einmaliges Teilen auf
  {
    pointList.add(edge1.point2);
    float reduceFactor=1;
    for (int i = 0; i < constructList.size(); i++)
    {
      edge1 = (Edge)constructList.get(i);
      edge1.makeNewEdge(0.8, reduceFactor*d, angle);
      reduceFactor = reduceFactor * verkuerzung;
    }
  }


  if ( type == 2)                                      //  Typ 2 ruft Methode für zweimaliges Teilen auf
  {
    //pointList.add(edge1.point2);
    float reduceFactor=1;
    for (int i=0; i < branchLength; i++)               //  Grad der Äste (entspricht der Anzahl Knicke zu Stamm zurück)
    {                                                  //  es wird jedoch nur eine der beiden neuen Kannten weitergeteilt
      if (i < constructList.size())
      {
        edge1 = (Edge)constructList.get(i);
        edge1.divide(0.7,reduceFactor *d,angle);
        reduceFactor = reduceFactor * verkuerzung;
      }
    }
    if (endPunktVariante == 1)
    {
      edge1.ending(PI/6);
    }
    if (endPunktVariante == 2)
    {
      edge1.ending2(punktRadius/2);
    }
  }
```

```
if (type == 3)                                                    // Typ 3 ruft Methode für zweimaliges Teilen auf
{
 pointList.add(edge1.point2);
 float reduceFactor=1;
 for (int i=0; i < branchLength; i++)                             // Grad des Äste (entspricht der Anzahl Knicke zum Stamm zurück)
 {
  if (i < constructList.size())                                   // es werden zwei Kannten der printList zweigeteilt
  {
   edge1 = (Edge)constructList.get(i);
   edge1.divide(0.7,reduceFactor *d,angle);
   reduceFactor = reduceFactor * verkuerzung;
  }
  if (i < constructList.size()-1){
   edge1 = (Edge)constructList.get(i+1);
   edge1.divide(0.7,reduceFactor *d, angle);
  }
 }
 if (endPunktVariante == 1)
 {
  edge1.ending(PI/6);
 }
 if (endPunktVariante == 2)
 {
  edge1.ending2(punktRadius/2);
 }
}
```

```
if ( type ==4 || type ==5)                                        // Typ 4 und 5 rufen Methoden für zweimaliges Teilen auf
{
 pointList.add(edge1.point2);
 float reduceFactor=1;
 for (int i=0; i < branchLength; i++)                             // Grad des Äste (entspricht bei TYP 5 der Anzahl Knicke zum Stamm zurück)
 {
  int ll = constructList.size();                                  // Pro Teilungsdurchgang werden so viele Kannten geteilt wie die constructList zu BEGINN Kannten hat
  for (int j = 0; j < ll; j++)
  {
   float angle2 = random(angle,angle+PI/16);
   if (type == 4)
   {
    edge1 = (Edge)constructList.get(j);                           // Typ 4 nimmt immer an einer Fortlaufenden Stelle der constructList eine Kannte zum teilen
   }                                                              // unregelmässiges wachstum, Äste der schon geteilten Kannte können weitergeteilt werden
   else {                                                         // Hirarchie von Grad der Äste wird gebrochen
    edge1 = (Edge)constructList.get(0);                           // Typ 5 nimmt immer an der ersten Stelle der constructList eine Kannte zum teilen
   }                                                              // regelmässiges wachstum
   edge1.divide(0.7,reduceFactor*d, angle2);
   reduceFactor = reduceFactor * (verkuerzung+0.04);
  }
 }
```

```
      if (endPunktVariante == 1)
      {
        edge1.ending(PI/6);
      }
      if (endPunktVariante == 2)
      {
        edge1.ending2(punktRadius/2);
      }
    }
  }

}



class Edge
{
  Point point1;                                                    // Startpunkt
  Point point2;                                                    // Endpunkt
  float thickness = 0;
  float edgeAngle;

  Branch myBranch;


  Edge(Point pa, Point pb, float t)
  {
    point1 = pa;
    point2 = pb;
    thickness = t;

    edgeAngle = atan((pa.y-pb.y)/(pa.x-pb.x));
  }


  Edge(Point pa, Point pb, float t, Branch newBranch)
  {
    point1 = pa;
    point2 = pb;
    thickness = t;
    myBranch = newBranch;

    edgeAngle = atan((pa.y-pb.y)/(pa.x-pb.x));                     // Winkel der Kante wird berechnet
    if (edgeAngle > 0 && point1.x > point2.x)                      // Korrigiert Winkel, da atan nur Winkel zwischen + - PI liefert.
    {
      edgeAngle = edgeAngle - PI;
    }
    if (edgeAngle < 0 && point1.y < point2.y && point1.x > point2.x)
    {
```

```
    edgeAngle = edgeAngle + PI;
  }
}


//————————————————METHODEN————————————————————————————————

//———einmaliges Teilen der Kannte

void makeNewEdge(float l, float d, float winkel)                    // wird Benutzt vom „Baumtyp 1“
{
  float tV = l;
  Edge e;
  Point tP = new Point(point1.x+tV*(point2.x-point1.x),point1.y+tV*(point2.y-point1.y));          // neuer Punkt auf der Kannte im Teilverhältnis TV wird erstellt

  thickness = thickness * schwindFaktor;

  float range = winkel;
  float w = edgeAngle + random(-range,range);                       // Winkel des neuen Punktes variiert um Winkel der zu teilenden Kannte

    float newXPosition = tP.x +(cos(w)*d);                          //  neuer Punkt wird Berechnet
  float newYPosition = tP.y +(sin(w)*d);
  Point nPoint = new Point(newXPosition,newYPosition);
  if (nPoint.testAusgabeFenster () > 0)                             //  Methode wird aufgerufen, die überprüft, ob neuer Punkt im Ausgabefeld liegt.  1 = liegt drinn
  {
    e = new Edge(tP,nPoint, thickness, myBranch);                   //  Neue Kannte wird geschrieben und in constructlist für weitere Teilungen geschrieben
    myBranch.constructList.add(e);
  }
  else {
    nPoint = nPoint.intersection (tP);                              //  Schnittpunkt mit dem Ausgabefenster wird zurückgegeben
    e = new Edge(tP,nPoint, thickness, myBranch);                   //  Kannnte bis zum Rand des Ausgabefeldes wird geschrieben
  }
  myBranch.printList.add(e);
  myBranch.pointList.add(new Point(newXPosition,newYPosition));
}


//——————————ZWEIMALIGES TEILEN DIESER KANNTE————————————————

void divide(float teilVerhaeltnis,float d, float winkel)
{

  float range = winkel;
  float tV = teilVerhaeltnis;
  tV = random(tV,1);

  Point tP = new Point(this.point1.x+tV*(point2.x-point1.x),point1.y+tV*(point2.y-point1.y));       // neuer Punkt auf der Kannte im Teilverhältnis TV wird erstellt
  Edge e1 = new Edge(point1,tP,thickness, myBranch);                //  neue Teilstrecken auf der Kannnte werden definiert
```

```
Edge e2 = new Edge(tP,point2,thickness, myBranch);

Point tP2 = new Point(e1.point1.x+tV*(e1.point2.x-e1.point1.x),e1.point1.y+tV*(e1.point2.y-e1.point1.y));          // neuer Punkt auf der NEUEN Kannte im Teilverhältnis TV wird erstellt
Edge e3 = new Edge(e1.point1,tP2,thickness, myBranch);                                                          //  neue Teilstrecken auf der Kannnte werden definiert
Edge e4 = new Edge(tP2,e1.point2,thickness, myBranch);

thickness = thickness * schwindFaktor;                                                                          // Astdicke verkleinern

if (myBranch.type == 3 || myBranch.type == 4 || myBranch.type == 5)                 // Astdicke schneller verkleinern, weil bei diesen Typen pro Teilungszenario mehr Aeste entstehen
{
  if (endPunktVariante != 2)
  {
    thickness = thickness * schwindFaktor;
  }
}

float w = edgeAngle + range;
float newXPosition = tP.x +(cos(w)*d);                                                                          //  neuer Punkt wird Berechnet
float newYPosition = tP.y +(sin(w)*d);
Point newPoint = new Point(newXPosition,newYPosition);
int positionPunkt = newPoint.testAusgabeFenster();                                                              //  Methode wird aufgerufen, die überprüft, ob neuer Punkt im Ausgabefeld liegt.  1 = liegt drinn
if (positionPunkt==0)
{
  newPoint = newPoint.intersection (tP);                                                                        //  Schnittpunkt mit dem Ausgabefenster wird zurückgegeben
}
Edge e5 = new Edge(tP,newPoint, thickness, myBranch);                                                           //  Neue Kannte wird geschrieben

w = edgeAngle - range;
newXPosition = tP2.x +(cos(w)*d);                                                                               //  neuer Punkt wird Berechnet
newYPosition = tP2.y +(sin(w)*d);
newPoint = new Point(newXPosition,newYPosition);
int positionPunkt2 = newPoint.testAusgabeFenster();                                                             //  Methode wird aufgerufen, die überprüft, ob neuer Punkt im Ausgabefeld liegt.  1 = liegt drinn
if (positionPunkt2==0)
{
  newPoint = newPoint.intersection (tP2);                                                                       //  Schnittpunkt mit dem Ausgabefenster wird zurückgegeben
}
Edge e6 = new Edge(tP2,newPoint, thickness, myBranch);

myBranch.constructList.remove(this);                                                                            //  Kannte die 2mal geteiltwurde wird aus printList und constructList entfernt
myBranch.printList.remove(this);

myBranch.printList.add(e3);                                                                                     //  zwei Teilstücke der inzwischen geteilten Kannte werden in die printList geschriebn
  myBranch.printList.add(e4);
if (endPunktVariante == 1 || endPunktVariante == 2)
{
  myBranch.printList.add(e2);
}
myBranch.printList.add(e5);
myBranch.printList.add(e6);
```

```
  if (positionPunkt + positionPunkt2 == 2)                    //  wenn Kannten im Ausgabefenster liegen:
  {                                                           //    - Neue Kannten werden in zufälliger reihenfolge in constructlist geschrieben
   float a = random(1,10);                                    //    - Typen 3 und 4 können daher einfacher variiert werden und Typ 5 zeichnet sich nicht symetrisch (schöner

   if (a > 5)
   {
    myBranch.constructList.add(e5);
    myBranch.constructList.add(e6);
    myBranch.pointList.add(e5.point2);
    myBranch.pointList.add(e6.point2);
   }
   else {
    myBranch.constructList.add(e6);
    myBranch.constructList.add(e5);
    myBranch.pointList.add(e6.point2);
    myBranch.pointList.add(e5.point2);
   }
  }
  else                                                        //  nur Kannte die nicht bis ans Ausgabefeld grenzen werden in constructList zum Weiterteiln geschrieben
  {
   if (positionPunkt==1)
   {
    myBranch.constructList.add(e5);
    myBranch.pointList.add(e5.point2);
   }
   if (positionPunkt2==1)
   {
    myBranch.constructList.add(e6);
    myBranch.pointList.add(e6.point2);
   }
  }
}


//————————ERSTELLT KLEINE ZWEIGE AM ENDE DER GABELUNGEN     Blatttyp 1

void ending (float winkel)                                    //  schreibt zwei kleine Zweige am Ende der Gabelungen (pointList)
{                                                             //  speicher diese in endingList
  float w = winkel;
  float l = d/4;

  for (int i = 0; i < myBranch.pointList.size(); i++)
  {
   Point start = (Point)myBranch.pointList.get(i);
   int l2 = int(random(l/2,l));
   Point nP = new Point(start.x-l2/3,start.y-l2);
   Edge e = new Edge(start,nP,1,myBranch);
   myBranch.endingList.add(e);
```

```
    nP = new Point(start.x+l2/3,start.y-l2);
    e = new Edge(start,nP,1,myBranch);
    myBranch.endingList.add(e);
    l= l * (verkuerzung+0.02);
   }
  }

 //————————ERSTELLT KLEINEN ZWEIG AM ENDE DER GABELUNG     Blatttyp 2

   void ending2 (float laenge)                                          //  schreibt einen kleinen Zweig am Ende der Gabelungen (pointList)
  {                                                                     //  speicher diesen in endingList
   float l = laenge;
   for (int i = 0; i < myBranch.pointList.size(); i++)
   {
    Point start = (Point)myBranch.pointList.get(i);
    Point nP = new Point(start.x,start.y-l);
    Edge e = new Edge(start,nP,1,myBranch);
    myBranch.endingList.add(e);
    l= l * verkuerzung;
   }
  }


//————————ZEICHNET KANNTE

 void drawMe(int rot, int gruen, int blau)
 {
  int r = rot;
  int g = gruen;
  int b = blau;
  strokeWeight(thickness);
  stroke(r,g,b);
  line(point1.x,point1.y,point2.x,point2.y);
 }


 //———ZEICHNET ELLIPSE————————————

 void drawMe(float ellipseX, float ellipseY, int rot, int gruen, int blau, int nummerDerAusfuerung)        //     linie mit Endpunkt
 {
  int r = rot;
  int g = gruen;
  int b = blau;
  int nDA = nummerDerAusfuerung;                                        // steuert Verkleinerung der Ellipse
  strokeWeight(thickness);
  stroke(r,g,b);
  fill(255);
```

```
    float rFactor;
    if (myBranch.type < 3)                                  //  Baumtypen 3, 4 und 5 müssen langsamer verkleinern
    {
      rFactor = pow(verkuerzung,nDA);
    }
    else {
      rFactor = pow(verkuerzung,nDA/3);
    }
    ellipse (point2.x, point2.y,ellipseX*rFactor,ellipseY*rFactor);
  }


}



class Point
{
  float x = 0;
  float y = 0;

  Point(float posX, float posY)
  {
    x=posX;
    y=posY;
  }



//--------Methoden-----------------------------

//--------generieren eines neuen Punktes von diesem Punkt aus

  Point makePointIn(float w, float d)                       //  wird nicht benötigt, da immer von Kanntenwinkel abhängig
  {
    float newXPosition = x +(cos(w)*d);
    float newYPosition = y +(sin(w)*d);
    return new Point(newXPosition,newYPosition);
  }



//--------berechnet Abstand zweier Punkte

  float distanceTo(Point other)
  {
    float deltaX = this.x - other.x;
```

```
  float deltaY = this.y - other.y;

  return sqrt(deltaX*deltaX + deltaY*deltaY);
}
```

```
//————————Prüft ob sich nPunkt im Ausgabefeld befindet

int testAusgabeFenster ()                                   //  gibt 1 zurück falls Punkt im Ausgabefeld lieget, sonst 0
{
  int positionPunkt = 0;
  if (x > ausgabeXpos && x < fensterX-abstand)
  {
    if (y > abstand && y < fensterY - abstand)
    {
      positionPunkt = 1;
    }
  }
  return(positionPunkt);
}
```

```
//————————Gibt Schnittpunkt der Strecke teilPunkt-nPunkt mit dem AusgabeFenster zurück

Point intersection (Point teilPunkt)
{
  Point teilp = teilPunkt;
  float x1 = ausgabeXpos;                                   //  prüft 8 Felder ausserhalb des Ausgabefensters
  float x2 = fensterX-abstand;                              //  schneidet mit den Geraden des Fensters, wenn nötig mit beiden
  float y1 = abstand;                                       //  Punkt ausserhalb des Fensters
  float y2 = fensterY - abstand;                            //  linke Kannte des Fensters
  float x3 = 0;                                             //  rechte Kannte des Fensters
  float y3 = 0;                                             //  obere Kannte des Fensters
                                                            //  untere Kannte des Fensters
  if (x < x1)                                               //  x-Koordinate des Schnittpunktes mit dem Ausgabefenster
  {                                                         //  y-Koordinate des Schnittpunktes mit dem Ausgabefenster
    if (y < y1)
    {
      x3 = x1;
      y3 =  teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
      if (y3 < y1)
      {
        y3 = y1;
        x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
      }
    }
```

```
  if (y > y2)
  {
    x3 = x1;
    y3 =  teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
    if (y3 > y2)
    {
      y3 = y2;
      x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
    }
  }

  if (y > y1 && y < y2)
  {
    x3 = x1;
    y3 =  teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
  }
}


if (x > x2)
{
  if (y < y1)
  {
    x3 = x2;
    y3 = teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
    if (y3 < y1)
    {
      y3 = y1;
      x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
    }
  }

  if (y > y2)
  {
    x3 = x2;
    y3 = teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
    if (y3 > y2)
    {
      y3 = y2;
      x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
    }
  }

  if (y > y1 && y < y2)
  {
    x3 = x2;
    y3 = teilp.y + (x3 - teilp.x)*(y-teilp.y)/(x-teilp.x);
  }
}
```

```
    if (x > x1 && x < x2)
    {
      if (y < y1)
      {
        y3 = y1;
        x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
      }
      if (y > y2)
      {
        y3 = y2;
        x3 = teilp.x + (x-teilp.x)/(y-teilp.y)*(y3 - teilp.y);
      }
    }

    Point nP =  new Point(x3,y3);                                   //  gibt Schnittpunkt zurück
    return (nP);
  }

}




//————————FIXES LAYOUT————————————

void drawLayoutFix()                                                // wird im setup aufgerufen
{
  smooth();

  fill(162, 195, 37);
  PFont font;
  font = loadFont(„CorporateSBQ-Light-100.vlw“);
  textFont(font,29);
  text(„cobaea scandens“, 40,58);


  font = loadFont(„CorporateSBQ-Bold-48.vlw“);
  textFont(font,18);
  pushMatrix();
  translate(84,184);
  rotate(PI/4.3);
  text(„neu“, 0,0);
  popMatrix();

  pushMatrix();
  translate(43,533);
  rotate(PI/4.3);
  text(„sichern“, 0,0);
```

```
popMatrix();

pushMatrix();
translate(153,606);
rotate(1.83*PI);
text(„quit“, 0,0);
popMatrix();

font = loadFont(„CorporateSBQ-Regular-48.vlw“);
textFont(font,18);
pushMatrix();
translate(85,291);
rotate(PI/4.3);
text(„typ“, 0,0);
popMatrix();

pushMatrix();
translate(57,355);
rotate(PI/4.3);
text(„anzahl“, 0,0);
popMatrix();


pushMatrix();
translate(52,430);
rotate(PI/4.3);
text(„vorlage“, 0,0);
popMatrix();


pushMatrix();
translate(161,257);
rotate(1.83*PI);
text(„grösse“, 0,0);
popMatrix();


pushMatrix();
translate(162,344);
rotate(1.83*PI);
text(„blätter“, 0,0);
popMatrix();


pushMatrix();
translate(164,420);
rotate(1.83*PI);
text(„winkel“, 0,0);
popMatrix();
```

```
  pushMatrix();
  translate(168,505);
  rotate(1.83*PI);
  text(„verkürzung“, 0,0);
  popMatrix();

  stroke(162, 195, 37);

  strokeWeight(4);
  line(100,715,136,608);

  strokeWeight(3);
  line(121,654,102,579);

  strokeWeight(2);
  line(113,623,152,505);
  line(131,567,107,470);
  line(120,518,147,420);
  line(133,465,106,385);
  line(124,433,146,345);
  line(134,387,114,316);
  line(123,347,145,259);

  strokeWeight(1);
  line(139,282,122,208);
  line(132,254,179,63);
  line(186,35,194,0);

}


//————–variables Layout————————

void drawLayoutChangeable() {                                   // wird im void draw aufgerufen
  smooth();


  fill(150,132,125);
  stroke(104,70,58);
  strokeWeight(1.1);

  ellipse(122,208,11,11);                                       // button neu

  ellipse(114,313,11,11);                                       // button typ
  ellipse(108,388,11,11);                                 // button anzahl
  ellipse(107,470,11,11);                                 // button vorlage

  ellipse(102,579,11,11);                                 // button sichern
```

```
ellipse(145,259,11,11);                                  // button grösse
ellipse(146,345,11,11);                                  // button blätter
ellipse(147,420,11,11);                                  // button winkel
ellipse(152,505,11,11);                                  // button verkürzung

ellipse(136,608,11,11);                                  // button quit


noStroke();
fill(0);
rect(28,60,110,65);

if (mouseX > 85 && mouseX < 130)
{
  if (mouseY > 175 && mouseY < 220)
  {
    fill(250);
    noStroke();
    ellipse(122,208,11,11);                              // button neu
  }
}


if (mouseX > 85 && mouseX < 125)
{
  if (mouseY > 285 && mouseY < 325)
  {
    fill(250);
    noStroke();
    ellipse(114,313,11,11);                              // button typ
  }

}


if (mouseX > 55 && mouseX < 115)
{
  if (mouseY > 350 && mouseY < 400)
  {
    fill(250);
    noStroke();
    ellipse(108,388,11,11);                               // button anzahl
  }

}


  if (mouseX > 55 && mouseX < 120)
{
  if (mouseY > 425 && mouseY < 480)
```

```
  {
    fill(250);
    noStroke();
    ellipse(107,470,11,11);                                   // button vorlage
  }

}


if (mouseX > 140 && mouseX < 205)
{
  if (mouseY > 220 && mouseY < 270)
  {
    fill(250);
    noStroke();
    ellipse(145,259,11,11);                                   // button grösse
  }

}


if (mouseX > 138 && mouseX < 205)
{
  if (mouseY > 310 && mouseY < 355)
  {
    fill(250);
    noStroke();
    ellipse(146,345,11,11);                                   // button blätter
  }

}


if (mouseX > 138 && mouseX < 205)
{
  if (mouseY > 385 && mouseY < 430)
  {
    fill(250);
    noStroke();
    ellipse(147,420,11,11);                                   // button winkel
  }

}


if (mouseX > 144 && mouseX < 245)
{
  if (mouseY > 455 && mouseY < 515)
  {
    fill(250);
```

```
    noStroke();
    ellipse(152,505,11,11);                                   // button verkürzung
   }

  }


  if (mouseX > 45 && mouseX < 110)
  {
   if (mouseY > 520 && mouseY < 590)
   {
    fill(250);
    noStroke();
    ellipse(102,579,11,11);                                   // button sichern
   }

  }


  if (mouseX > 130 && mouseX < 182)
  {
   if (mouseY > 575 && mouseY < 620)
   {
    fill(250);
    noStroke();
    ellipse(136,608,11,11);                                   // button quit
   }

  }


  if (mouseX > 40 && mouseX < 250)
  {
   if (mouseY > 35 && mouseY < 65)
   {
    smooth();
    fill(200,170,150);
    noStroke();
    PFont font;
    font = loadFont(„CorporateSBQ-Regular-48.vlw“);
    textFont(font,11);
    text(„DWF PROCESSING“, 44,83);                            // button titel
    text(„DINA, KAJ, RENATE“, 44,99);
    text(„SS 07“, 44,115);
   }

  }

}
```