A bidirectional interface between BwinPro and GroIMP

Project report, University of Göttingen
Department: Ecoinformatics, Biometrics and Forest Growth

Tim Oberländer

December 10, 2019

Contents

Chapter 1

Introduction

1.1 Idea

BwinPro and GroIMP are both tools used on similar topics and each of them can do some things better or different, so it is useful to have a simple and customizable way to transfer data between them. This project is trying to create such a way.

1.1.1 Overview

This documentation is separated in three parts: a short introduction into the used software and concept, the transfer from GroIMP to BwinPro and the transfer from BwinPro to GroIMP. The second and the third chapters are separated into the same three sections: Documentation, Theory and Code. Mainly the documentation should be enough for the normal use. Also there is a small example for the export from GroIMP to BwinPro.

1.2 BwinPro

BwinPro is a stand-simulation software developed by the "Nordwestdeutsche forstliche Versuchsanstalten". It is mainly for analyzing and evaluating forest stands. The program can read in existing stand data as well as generate random stands. It can generate reports for example about the growth or the volume in 5 year simulation periods. [1]

It is open source and can be found online:
`https://www.nw-fva.de/?id=194`

1.3 GroIMP

GroIMP(Growth Grammar-related Interactive Modelling Platform) is a software for the algorithmic 3D modelling, especially used for plants. GroIMP uses growth grammars(a sort of parallel graph grammar) to simulate the growth of plants and trees.
This is realized by the programming language XL, which is an extension of Java.
Conditioned by the plugin based program structure, GroIMP offers a lot of additional functions like NURBS surfaces.[2]
GroIMP is open source as well and can be found online:
`https://sourceforge.net/projects/GroIMP/`

1.4 Storage

To transfer data it is necessary to have a basic understanding about the way data is stored on both platforms.

1.4.1 GroIMP

GroIMP stores data in a gsz file which is a normal zip archive.
It contains at least 3 files:

- **model.rgg**
  This file can have different names but it always contains the code used to create the 3D model.
- **project.gs**
  This file contains the basic set-up of the project including the GroIMP user interface configuration.
- **graph.xml**
  In this file the actual model is stored in a graph like the one which can be seen in GroIMP on /panels/2D/Graph.

1.4.2 BwinPro

BwinPro stores all trees in one huge XML file, this XML file is separated in different segments:

- basic information
- species in the stand
- corner marks to create the area where the trees are located
- the trees (each tree is defined by 25 parameters)

1.5 Basic Tree

For both platforms there is an minimal amount of data needed for each tree, otherwise it is not possible to create a stand. This means the core project transfers always the same two simple trees with the same parameters.

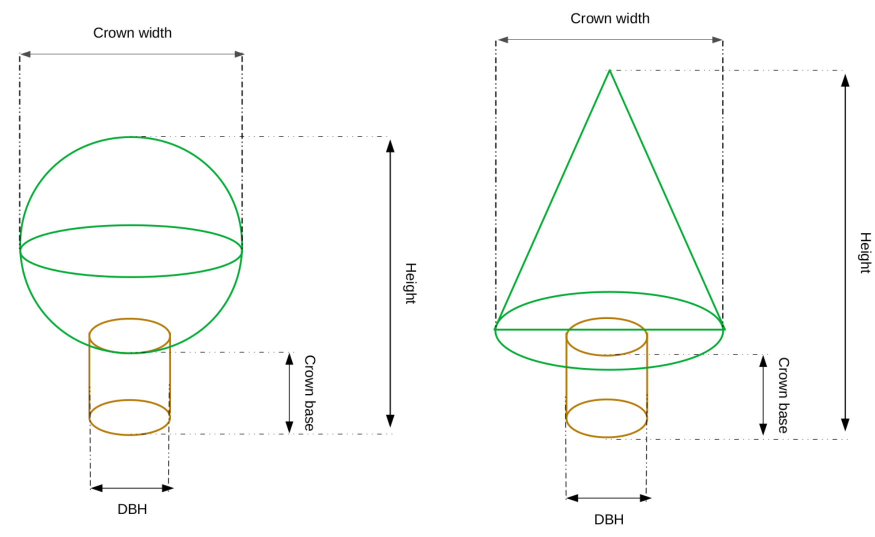


Figure 1.1: Simplified trees with basic measures(left:deciduous tree, right:conifer)

Besides the four parameters explained in Figure 1.1, the position, age and the species of the tree defined by a code are necessary
So the mandatory parameters are:

- x,y,z
- height
- dbh
- crown diameter
- crown base
- species code
- age

This data is enough to present a tree in BwinPro[1] as well as in GroIMP.

Chapter 2

GroIMP to BwinPro

The export from GroIMP to BwinPro is a little difficult, because there are a lot of ways to model a tree in GroIMP. Luckily GroIMP has got a lot of handy methods to analyse models.
For this reason the export function is implemented directly in GroIMP.
To generate an XML file is quite simple because XL is fully compatible to Java.

2.1 Documentation

2.1.1 Installation

A real installation is not necessary, only the testArea.gsz and the export.gsz files are needed, which are stored in "/GroIMP2Bwin/". Caused through the fact that these files need to be manipulated, it is recommended to create firstly a copy of each.

2.1.2 Basic concept

To run the export2bwinPro function in export.gsz two things are needed to be loaded into the RGG code:

- **the pointer(s):** they get created in the TestArea.
- **the stand:** it has to be slightly manipulated

2.1.3 TreePointer

The TreePointer is a XL module that defines what a tree is and the measures of the tree.
It gets this information out of calculations, which should work for most tree-models, but not for all.
For this reason there is the TestArea to check if the tree-model works with the pointers.

TestArea

The TestArea.gsz uses the same pointers as the Export.gsz, but the TestArea has got a test button that draws the simplified tree around the tested tree-model to check if the pointer works well.
For this test it is necessary to add one tree (with a pointer) of each species in the stand to the TestArea init() function,
for example:

```
Axiom==> [TreePointer(Picea,15)Translate(2,4,0)YourFancyTree()];
```

The TreePointer needs two parameters: the speciesCode and the age.
The codes (in this case for Picea) are on top of the Export.gsz and the TestArea.gsz files.
Instead of "Translate()" it is also possible to use any other function to set the position.
After hitting the test button the result should look nearly like this(Figure 2.1):

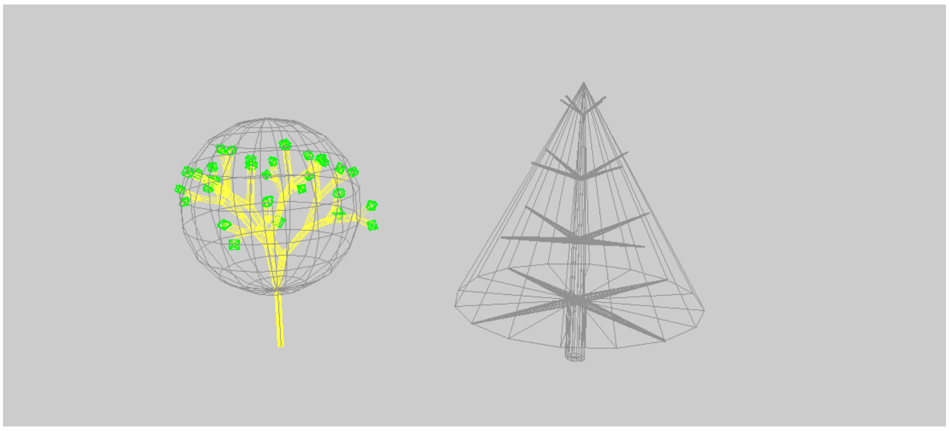


Figure 2.1: Example result from the TestArea.gsz

For simplification the crowns of the deciduous trees are always spheres.
If the simplified trees are looking well the modify part can be skipped.

Modify

Now it depends on the question if all trees do not match with the simplified trees or just some of them:

Nothing fits

In this case the calculation functions don't work on the correct way. In most cases it is obvious which does not work properly and has to be replaced. Some ideas why it could go wrong:

- The objects are not instances of a subclass of ShadedNull
- The crown is modelled by just one object.
- The stem is neither a cylinder nor a frustum.

Some fit
When trees are modelled in different ways it is possible that the original pointer only works for some of them.
In this case there is the possibility to create a second (or more) pointer for the kind of tree that does not work with the original pointer in a style like this:

```
module MyPinusPointer(int code, int age) extends TreePointer(code,age)
```

Now the not working calculations can be overwritten in the new pointer and fixed in the same way as in the "nothing fits" chapter above.

2.1.4 The stand

Edit

For the export it is necessary that all trees have a pointer like this:

```
[TreePointer(code,age) yourTree...]
```

In case of multiple pointers the pointers have to be the same as in the TestArea.
In case of growing trees something like this will be useful for the age:

```
        p:TreePointer::>{p.age++;}
```

2.1.5 Run

There are some parameters to customize the export but only two of them are necessary: "scale" (in case the stand is not in 1:1) and "saveDir", the place the exported file will be stored. Now after saving there should be a button with the label "export2BwinPro". If everything is working fine the GroIMP-console should say: 'File saved'.

2.2 Theory

2.2.1 Basic export

As written in the introduction, the basic transfer works with a tree defined by a vector of 9 components.

So the code needs to read these values out of the GroIMP stand and also it must detect what is a tree and what not.

2.2.2 Step by Step

Start

A stand out of 3 trees and one stone in GroIMP can be imagined like the Example in figure 2.2.

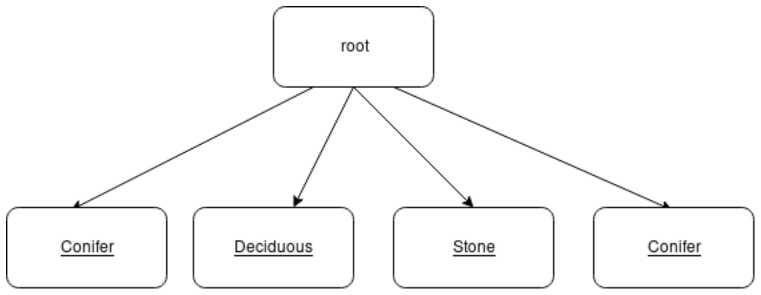


Figure 2.2: A simplified graph of a small stand

Measures

With a basic knowledge in GroIMP/XL it is possible to read all the measures and the location directly out of the model(figure 2.3).

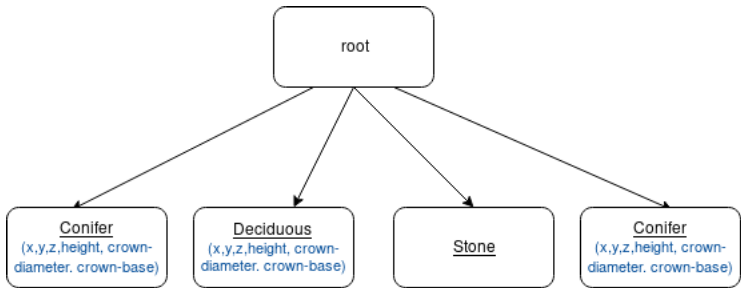

root
Conifer
(x,y,z,height, crown-diameter. crown-base)
Deciduous
(x,y,z,height, crown-diameter. crown-base)
Stone
Conifer
(x,y,z,height, crown-diameter. crown-base)

Figure 2.3: Step 1: Calculations of the measures and locations of the trees

Detecting a tree

GroIMP handles the whole model as one graph so it is not directly clear what a tree is and where it starts. For this problem the project uses pointers which are basically just saying the sub-graph under the pointer is a tree(figure 2.4).

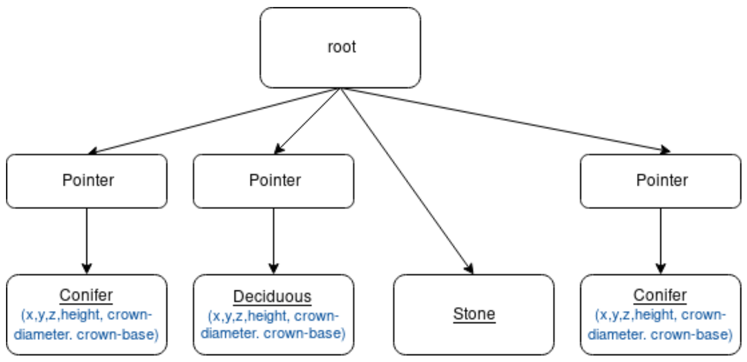

root
Pointer
Pointer
Pointer
Conifer
(x,y,z,height, crown-diameter. crown-base)
Deciduous
(x,y,z,height, crown-diameter. crown-base)
Stone
Conifer
(x,y,z,height, crown-diameter. crown-base)

Figure 2.4: Step 2: Using pointers to indicate trees

Age and species

Now there are only 2 values missing: The age and the speciesCode (type), both can not be read out of the tree model(Figure 2.5). They must be added directly as parameters to the TreePointer

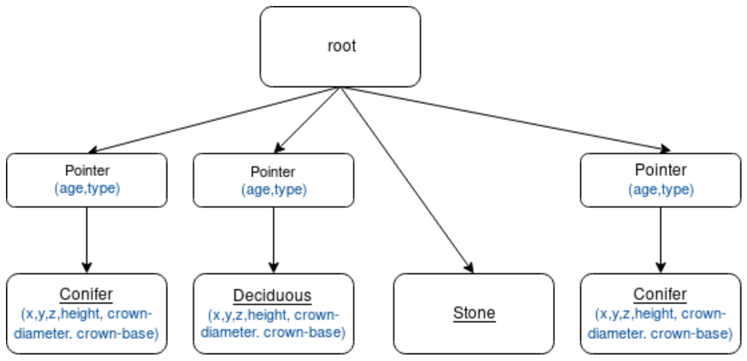

root
Pointer
(age,type)
Pointer
(age,type)
Pointer
(age,type)
Conifer
(x,y,z,height, crown-diameter. crown-base)
Deciduous
(x,y,z,height, crown-diameter. crown-base)
Stone
Conifer
(x,y,z,height, crown-diameter. crown-base)

Figure 2.5: Step 3: Adding missing parameters to the pointers

Bringing it together

The pointer contains not only the values for the age and the speciesCode, it also got "get-functions" for all the other 23 parameters.
But only the functions for the seven values (shown above) are real calculations, while the other functions are just returning fixed values.
In this way the export script only has to walk through all pointers and to write the results in an XML-file.
This way also gives the possibility to add new calculations by simply overwriting the old function.

2.3 Code

2.3.1 Overview

There are 2 gsz files needed to export a GroIMP stand to BwinPro:

- **TestArea.gsz:** to set up the right pointer for each species
- **Export.gsz:** bring it all together and print it to a XML-file

2.3.2 TestArea

There are many ways to model a tree/stand in GroIMP. Based on this there is always the possibility that the pointer doesn't work properly. To check if the pointer works the TestArea uses the pointer to draw a simplified tree (similar to the one imported from BwinPro to GroIMP)

```
public void test()
[
//deciduous
p:TreePointer,(p.getCode()<500)==>[
        Translate(p.getX(),p.getY(),p.getZ())
        Cylinder(p.getCrownBase()+p.getCrownDiameter()/2,p.getDBH()
            /200)
        Sphere(p.getCrownDiameter()/2)
];
//conifers
p:TreePointer,(p.getCode()>=500)==>[
        Translate(p.getX(),p.getY(),p.getZ())
        Cylinder(p.getCrownBase(),p.getDBH()/200)
        Cone(p.getHeight()-p.getCrownBase(),p.getCrownDiameter()/2)
];
]
```

2.3.3 Export

Depending on the design of the pointer the export is quite simple: the function just walks through all the pointers and writes the result to an XML-file.

Walk Trough

GroIMP comes with the possibility to walk through all instances of a module (including the modules extending the original modules).
So it is quite simple to get all trees and so all their getter functions.

```
int i=0;
private void createTrees()
[
        b:TreePointer::>{
                createTree(b,i);
                i++;
        }
]
```

XML writer

The XML writer is normal Java code using the "javax.xml" library.

There are still some values hard coded, but only about the area. These hardcoded values are defining the shape and the slope of the area and the meters above sea level.

2.4 Example

A simple step by step example how to export a single tree from GroIMP to BwinPro.

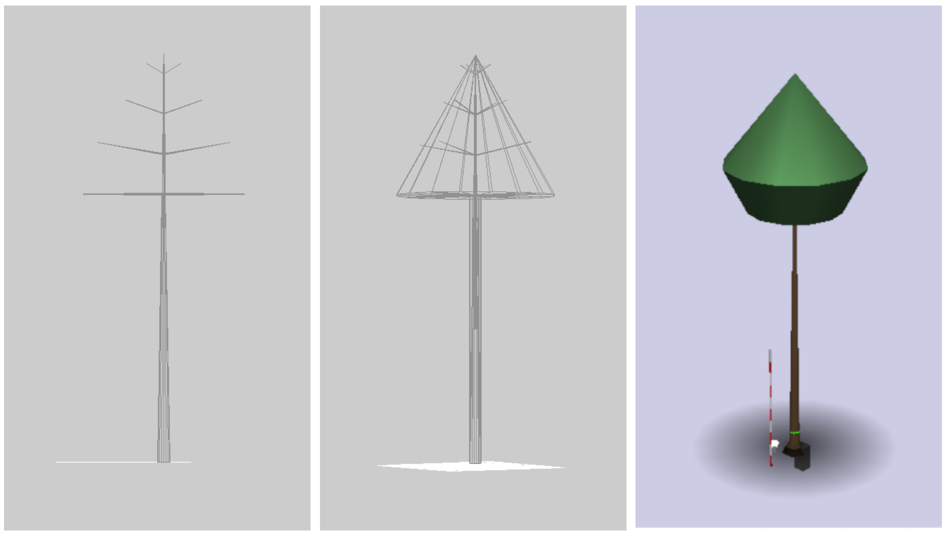


Figure 2.6: a simple conifer || tree with test surface || same tree in BwinPro

2.4.1 A simple conifer

The following code creates the tree shown in figure 2.6 and the graph shown in figure 2.7. It is a really simple construction of frustums and was only designed for this example.

```

protected void init ()[
Axiom ==> [Translate(3,4,0)
        [Frustum(20,0.46,0.13)
        [Rotate(90,0,0)Frustum(6,0.07,0.01)]
        [Rotate(0,90,10)Frustum(6,0.07,0.01)]
        [Rotate(90,180,0)Frustum(6,0.07,0.01)]
        [Rotate(90,270,0)Frustum(6,0.07,0.01)]

        Frustum(3,0.13,0.1)
        [Rotate(80,0,0)Frustum(5,0.05,0.01)]
        [Rotate(0,80,0)Frustum(5,0.05,0.01)]
```

```
    [Rotate(100,180,0)Frustum(5,0.05,0.01)]
        [Rotate(0,280,0)Frustum(5,0.05,0.01)]

        Frustum(3,0.1,0.07)
        [Rotate(70,0,0)Frustum(3,0.03,0.01)]
        [Rotate(0,70,0)Frustum(3,0.03,0.01)]
        [Rotate(110,180,0)Frustum(3,0.03,0.01)]
        [Rotate(0,290,0)Frustum(3,0.03,0.01)]

        Frustum(3,0.07,0.04)
        [Rotate(60,0,0)Frustum(1.5,0.01,0.01)]
        [Rotate(0,60,0)Frustum(1.5,0.01,0.01)]
        [Rotate(120,180,0)Frustum(1.5,0.01,0.01)]
        [Rotate(0,300,0)Frustum(1.5,0.01,0.01)]
        Frustum(1.5,0.04,0)
    ]
];
]
```

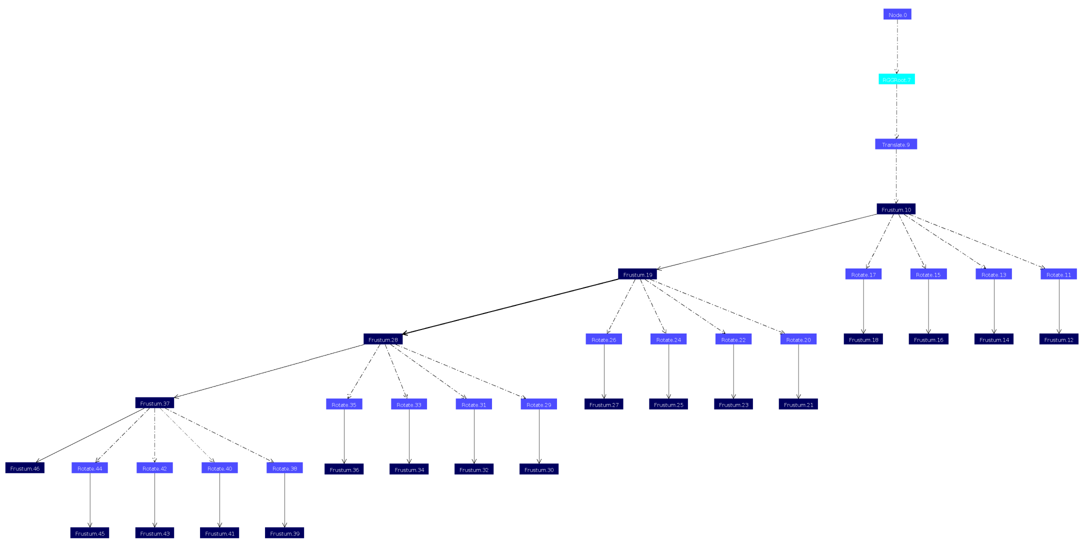


Figure 2.7: The graph of the simple conifer

2.4.2 Add to TestArea

To create and test the TreePointer the simple conifer gets copied to the TestArea.gsz file.
Also the pointer is added to the tree in line 72.

```
//Species Codes
int Abies=511;
int Acer=321;
int Alnus=421;
int Betula=412;
int Fagus=211;
int Fraxinus=311;
int Larix=811;
int Picea=511;
int Pinus=711;
int Pseudosuga=611;
int Quercus=110;
int Quercus_rubra=113;
int Sorbus=357;

const float scale=1; // don't mess with this!!!!!


module TreePointer(int code, int age){
        float getBHD(){
                        float bhd=max((*this-minDescendants-> Cylinder
                            *).getRadius());
                        if(bhd!=Float.NEGATIVE_INFINITY){
                                return 200*bhd*scale;
                        }else
                        bhd=max((*this-minDescendants-> Frustum*).
                            getBaseRadius());
                        if(bhd!=Float.NEGATIVE_INFINITY){
                                return 200*bhd*scale;
                        }
        }

        float getCrownDiameter(){
                float xmin=(min(endlocation((*this -descendants->
                    ShadedNull*)).x)-this.getX())*-1;
                float xmax=max(endlocation((*this -descendants->
                    ShadedNull*)).x)-this.getX();
                float ymin=(min(endlocation((*this -descendants->
                    ShadedNull*)).y)-this.getY())*-1;
                float ymax=max(endlocation((*this -descendants->
                    ShadedNull*)).y)-this.getY();
                return((xmin+xmax+ymin+ymax)/2)*scale;
        }
        float getCrownBase(){
                return max(endlocation((*this-minDescendants->
                    ShadedNull*)).z)*scale;
        }
        float getHeight(){
                return max(endlocation((*this -descendants->
                    ShadedNull*)).z)*scale;
        }


        int getAge(){return age;}
        int getCode(){return code;}
        float getX(){return min(location((*this -minDescendants->
            ShadedNull*)).x)*scale;}
        float getY(){return min(location((*this -minDescendants->
            ShadedNull*)).y)*scale;}
        float getZ(){return min(location((*this -minDescendants->
            ShadedNull*)).z)*scale;}

        //just in case some one wanna have fun, works with this
            settings...
        boolean isAlive(){return true;}
        boolean isTakenOut(){return false;}
        int getTakeOutMonth(){return 0;}
        int getTakeOutYear(){return -1;}
        String getTakeOutReasion(){return "anderer";}
        int getSiteIndex(){return 0;}
        boolean isZBaum(){return true;}
        boolean isZBaumTmp(){return false;}
        boolean isHabitatBaum(){return false;}
        int getKraftscheKlasse(){return 0;}
        int getSchicht(){return 1;}
        int getFlaechenFaktor(){return 1;}
        float getVolume_cbm(){return 0.0;}
        float getVolume_cbm_dead(){return 0.0;}
}


protected void init ()
[
        Axiom ==> TreePointer(611,60)[Translate(3,4,0)
                        [Frustum(20,0.46,0.13)
                        [Rotate(90,0,0)Frustum(6,0.07,0.01)]
                        [Rotate(0,90,10)Frustum(6,0.07,0.01)]
                        [Rotate(90,180,0)Frustum(6,0.07,0.01)]
                        [Rotate(90,270,0)Frustum(6,0.07,0.01)]

                        Frustum(3,0.13,0.1)
                        [Rotate(80,0,0)Frustum(5,0.05,0.01)]
                        [Rotate(0,80,0)Frustum(5,0.05,0.01)]
                        [Rotate(100,180,0)Frustum(5,0.05,0.01)]
                        [Rotate(0,280,0)Frustum(5,0.05,0.01)]

                        Frustum(3,0.1,0.07)
                        [Rotate(70,0,0)Frustum(3,0.03,0.01)]
                        [Rotate(0,70,0)Frustum(3,0.03,0.01)]
                        [Rotate(110,180,0)Frustum(3,0.03,0.01)]
                        [Rotate(0,290,0)Frustum(3,0.03,0.01)]

                        Frustum(3,0.07,0.04)
                        [Rotate(60,0,0)Frustum(1.5,0.01,0.01)]
                        [Rotate(0,60,0)Frustum(1.5,0.01,0.01)]
                        [Rotate(120,180,0)Frustum(1.5,0.01,0.01)]
                        [Rotate(0,300,0)Frustum(1.5,0.01,0.01)]
                        Frustum(1.5,0.04,0)
        ]

        ];
]

public void test()
[
        p:TreePointer,(p.getCode()<500)==>[Translate(p.getX(),p.getY()
            ,p.getZ())Cylinder(p.getCrownBase()+p.getCrownDiameter()
            /2,p.getBHD()/200)Sphere(p.getCrownDiameter()/2)];
        p:TreePointer,(p.getCode()>=500)==>[Translate(p.getX(),p.getY
            (),p.getZ())Cylinder(p.getCrownBase(),p.getBHD()/200)Cone(
            p.getHeight()-p.getCrownBase(),p.getCrownDiameter()/2)];

]
```

The test result can be seen in figure 2.6(middle) and the graph with the Pointer in figure 2.8.

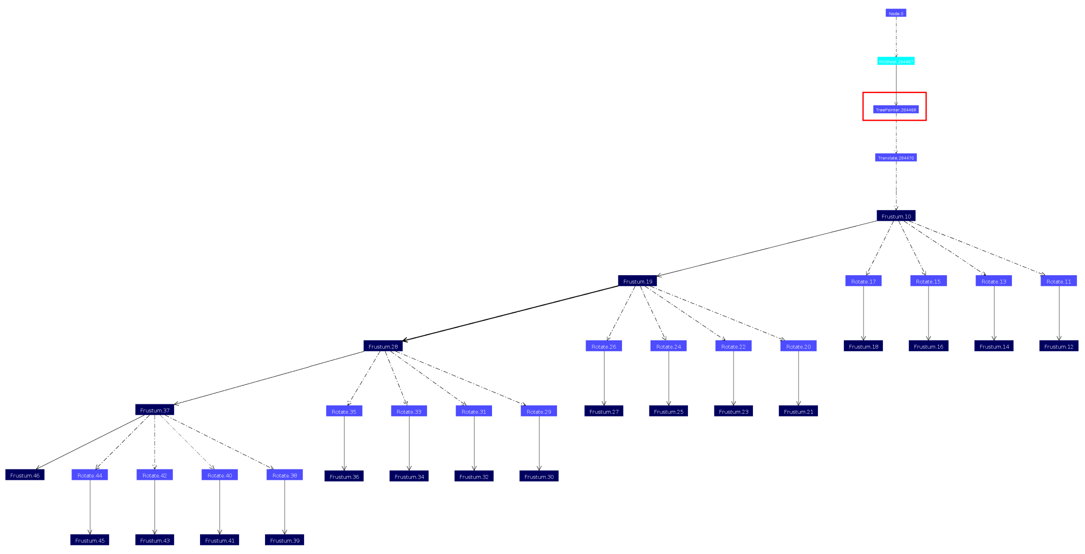


Figure 2.8: The graph of the simple conifer including the pointer(highlighted)

2.4.3 Export

The code from the TestArea(line 16 to 100) gets copied to the export gsz file.
After hitting the export button, a TreeGross-XML file will be generated at the given location(saveDir).
This file can be opened in BwinPro showing the result of figure 2.6(right).

Chapter 3

BwinPro to GroIMP

For importing TreeGross XML files to GroIMP, the BwinReader GroIMP-plugin is used, which is basically a XML parser writing a line of code for each tree, like this:

```
[Translate(x,y,z)]Tree_511(parameters...)]
```

3.1 Documentation

3.1.1 Install

For now this plugin is not a part of the original GroIMP, so it must be installed afterwards. But it can be found online:
https://sourceforge.net/p/groimp/code/HEAD/tree/branches/BwinReader/

To install the plugin the folder "BwinReader" has to be copied in the directory where your GroIMP code is stored and the Project has to be imported to Eclipse. [3]

3.1.2 Simple import

If the plugin is installed properly TreeGross XML files can simply be opened in the normal filechoose-dialog like in figure 3.1:

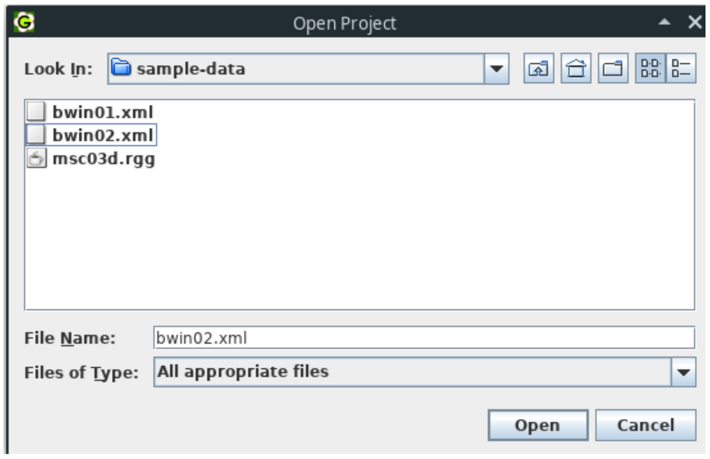


Figure 3.1: Open TreeGross XML

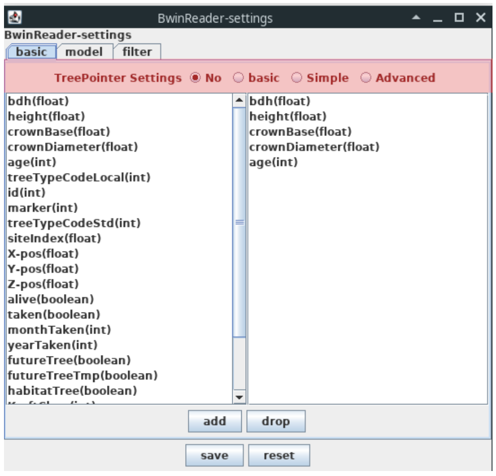
After opening the normal GroIMP RGG interface shows up with the generated RGG code in a temp-file.
It is necessary that the TreeGross XML file ends with ".xml" otherwise GroIMP cannot open it.

3.1.3 Settings

Before importing, the result can be customized with the setting panel, which can be found on:
/panels/BwinReader-settings.

TreePointer

The first setting option is the TreePointer that will get interesting if it is planed to transfer the stand back to BwinPro later(figure 3.2).



BwinReader-settings

TreePointer Settings [✓ No] basic Simple Advanced

```
bdh(float)                      bdh(float)
height(float)                   height(float)
crownBase(float)                crownBase(float)
crownDiameter(float)            crownDiameter(float)
age(int)                        age(int)
treeTypeCodeLocal(int)
id(int)
marker(int)
treeTypeCodeStd(int)
siteIndex(float)
X-pos(float)
Y-pos(float)
Z-pos(float)
alive(boolean)
taken(boolean)
monthTaken(int)
yearTaken(int)
futureTree(boolean)
futureTreeTmp(boolean)
habitatTree(boolean)
```

Figure 3.2: TreePointer settings(red) and parameter settings

Which type of pointer is needed depends on the way the trees will be modelled when they get transferred back to BwinPro.

- no: no pointer (no transfer back)
- basic: same pointer for all trees (all trees will be modelled in the same way)
- simple: one pointer for all deciduous trees and one for conifers (all deciduous trees /conifers will be modelled in the same way)
- advanced: different pointers for each species (trees are modelled in different ways)

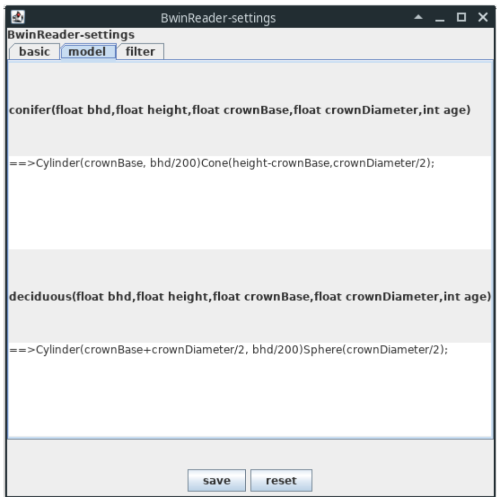
Parameters

BwinPro stores 25 parameters for each tree [1], by usually not all of them are needed in GroIMP. Also it depends on the project which of them are useful.
So the parameter setting provides the possibility to select from the pool of values(figure 3.2).

Models

After importing the trees are modelled in two different ways, one way for deciduous and the other for conifers.
These models can always be changed after importing. But for importing multiple stands, it could be useful to change these models by default.
Also if the parameter settings were changed, it is possible that the basic

models do not work any longer.
In this case it also would make sense to change the default model.



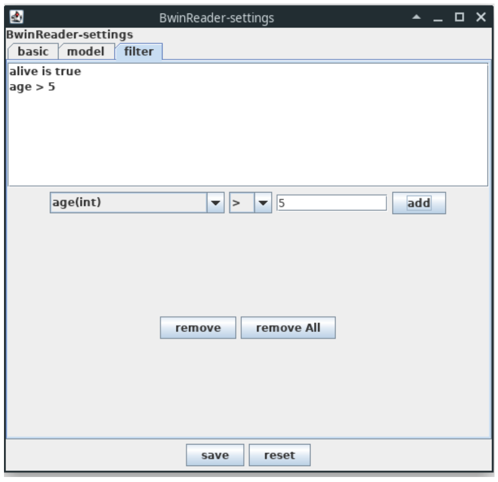
```
conifer(float bhd,float height,float crownBase,float crownDiameter,int age)
==>Cylinder(crownBase, bhd/200)Cone(height-crownBase,crownDiameter/2);

deciduous(float bhd,float height,float crownBase,float crownDiameter,int age)
==>Cylinder(crownBase+crownDiameter/2, bhd/200)Sphere(crownDiameter/2);
```

Figure 3.3: Model settings

Filter

This section was created for the case that only a part of the trees have to be imported, for example only trees older than 5 years(figure 3.4).



```
alive is true
age > 5
```

age(int) > 5

Figure 3.4: Filter settings

For boolean variables the operator does not matter. Only “true“ or “false“ must be entered in the value filed.

3.2 Theory

3.2.1 Simple import

A simple tree is defined as : $< x, y, z, h, dbh, cD, cB, spc, age >$
(All this data can easily be read out of the TreeGross XML file.)

A function
$f(bwinTreeVector) \rightarrow GroIMPTreeCode$
would look like this:
$Btree =< x, y, z, h, dbh, cD, cB, spc, age >$

$$f(Btree) = \begin{cases} [Translate(x, y, z)Cylinder(cB + cD/2, dbh/2)Sphere(cD/2)] & : spc \in deciduous \\ [Translate(x, y, z)Cylinder(cB, dbh/2)Cone(h - cB, cD/2)] & else \end{cases}$$

This function creates a result shown in figure 3.5:

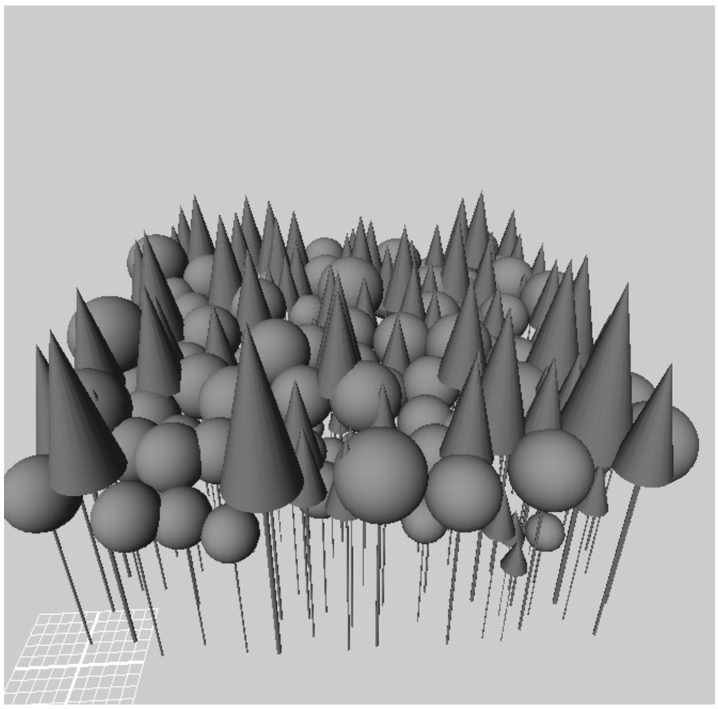


Figure 3.5: A first result of the theoretical import function

Even if the graphical result is quite suitable for a first concept, the code is not usable at all:

```

protected void init ()
[
Axiom ==>
[Translate(20.36,44.67,0.0)Cylinder(16.85,0.120749995)Sphere(2.85)]
[Translate(2.79,44.67,0.0)Cylinder(17.03,0.1082)Cone(7.289999,1.79)]
[Translate(11.98,44.33,0.0)Cylinder(15.66,0.0844)Cone(5.880001,1.535)]
[Translate(24.07,44.2,0.0)Cylinder(15.52,0.08279)Cone(5.779999,1.52)]
[Translate(29.47,43.96,0.0)Cylinder(15.175,0.09355)Sphere(2.425)]
...
```

In this code there is no way how tree models could be changed afterwards. To give this possibility the plugin creates modules for each kind of tree:

```

//Picea abies http://de.wikipedia.org/wiki/Gemeine_Fichte
module Tree_511(float height,float stemRadius, float crownBase, float
    crownRadius)==>Cylinder(crownBase,stemRadius)Cone(height-crownBase
    ,crownRadius);

//Quercus http://de.wikipedia.org/wiki/Eiche
module Tree_110(float height, float stemRadius, float crownBase, float
     crownRadius)==>Cylinder(crownBase+crownRadius,stemRadius)Sphere(
    crownRadius);


protected void init ()
[
Axiom ==>
[Translate(20.36,44.67,0.0)Tree_110(25.46,0.120749995,14.0,2.85)]
[Translate(2.79,44.67,0.0)Tree_511(24.32,0.1082,17.03,1.79)]
[Translate(11.98,44.33,0.0)Tree_511(21.54,0.0844,15.66,1.535)]
[Translate(24.07,44.2,0.0)Tree_110(21.3,0.08274999,15.52,1.52)]
...
```

Now changes can be done simply and separately for each species.

3.3 Code

3.3.1 Classes

- **BwinSourceFile:**
  This is basically the same file as grogra.pf.ui.registry.SourceFile except for one line where the input file gets replaced with the result of the Parser.parser() function.

- **Parser:**
  - creates temporary RGG file
  - initializes the content handler

  - runs the XML-Reader with an instance of XContentHandler
  - prints the result out of the XContentHandler in the RGG file

- **XContentHandler**
  - implements ContentHandler
  - creates an ArrayList with trees and one with the species

- **Tree**
  - Mainly a hashmap with the German names of the Bwin parameters and their values as strings
  - four toString functions (one for each treePointerMode), the result depends on the parameterHandler
  - a filter function that checks if this tree can pass the filter options

- **Species**
  - contains the basic information about the species (code, Latin name)
  - a toString function that returns the Latin name (and the link to the German Wikipedia page) and the module (out of the Parameter-Handler)
  - the model depends on the species code (codes lower than 500 are for conifers)

- **ParameterHandler**
  - organizes the reading and writing in java.util.pref.Preferences
  - contains an array with ParameterValues of all Bwin parameters
  - contains all values which can be changed in the setting mode:
    - pointer mode
    - conifer & deciduous model
    - selection of the parameters which will be imported
    - list of filters which are used
  - and a reset function to set all values to the basic setting

- **ParameterValue**
  basically an abstract data type with 3 values: German-name (for the hashmap), English-name (for the interface) and the value.

- **FilterValue**
  a combination of the ParameterValue, the operator to compare and the value to compare

- **GuiLauncher**
  runs the GUI
- **SettingGui**
  basically an interface for the ParameterHandler

3.3.2 Theoretical walk through

A really basic walk through, what happens after hitting "open" in the file chooser:

1. the BwinSourceFile runs like grogra.pf.ui.registry.SourceFile until the parser starts
2. **Parser(parseToTmpFile):**
   (a) runs the parse function with the link to the opened file
   (b) **Parser(parse):**
      i. initializes the basic XML-Parser
      ii. runs XML-Parser with the XContentHandler
      iii. **XContentHandler:**
         A. writes the values of each species in an instance of "Species" and stores them as an ArrayList.
         B. writes the values of each tree in a hash map and stores this hash map in an instance of the Tree class
         C. checks with the filters function of tree if the tree should be stored and if so, stores it in an array List.
      iv. walks through the results of XContentHandler and writes the XL-code
      v. the Tree.printTree function gives the parameters back depending on the ParameterHandler
   (c) creates a tmp RGG file and writes the result in it
3. opens the result like a normal RGG file in GroIMP

3.3.3 InitX problem

The imported code will have multiple initX (init1, init2. . .) functions.
This solution is necessary because Java can not handle functions with more than around 2000 lines of code. So the init function has to be split into several functions which are combined like a chain.

Bibliography

[1] "ForestSimulator 7.5 - English Manual" - Prof. Dr. Jürgen Nagel (Northwest German Forest Research Station)

[2] `http://wwwuser.gwdg.de/~GroIMP/grogra.de/software/GroIMP/index.html`

[3] "Entwicklung eines Plugins für die 3D-Modellierungsplatform GroIMP" - Dominick Leppich
`http://134.76.18.36/wordpress/wp-content/uploads/2018/02/plugin_entwicklung-1.pdf`