Data visualization in an immersive Virtual Reality environment

Alex Olwal 2000
alex@harmonic.se

Background

This project is part of VRGobi, a Dynamic Statistical Data Exploration project (See Appendix A), and is targeted for use in an immersive Virtual Reality environment supported by VRJuggler, a hardware API. The objective of the project is to enhance the visualization of a particular data set by tying it to a model of its real world representation. The classical data set that was chosen consists of six measurements on three species of the flea beetle (See Appendix B). The goal is to produce a complement to the three-dimensional point cloud that normally is used to represent the data. The idea is to project the point representation onto the floor, model every point as a beetle, which possess properties according to the measurements in the data set. It is also desirable to have the beetles move around and allow interaction with them.

Figure 1. Flea beetle.

Table of Contents

The application

The application works in such a way that when VRGobi is running and all the points are being drawn in space, they are also projected down onto the floor, where they are modelled as beetles. These beetles have the same body color as their corresponding point and their physics are a mapping from the statistical data that is represented by the point. At each time step, a counter is increased, determining where the beetle is drawn next and how it is rotated. The counter is also used for the motion, the animation, of the beetle. Thus all beetles are moving in random paths on the floor, seemingly in disorder. As one spot a beetle of interest, it can be pointed at with the wand and if clicked, it will be selected. Both the selected beetle and its corresponding point will be emphasized.

Research

In order to visualize the flea beetles, the first objective was to find appropriate sketches of the specific species. This was a lot harder than expected, due to the large amount of flea beetle species and the assumingly not so significant role of a particular one. Several instances were consulted – Parks Library and the Entomology department at Iowa State University, Ames Public Library as well as several resources on the Internet. The best biometric image found was a sketch from an old French book, see figure 2. Unfortunately only the top view was available, and only on one of the three species. The found picture was used as a basis for the design.

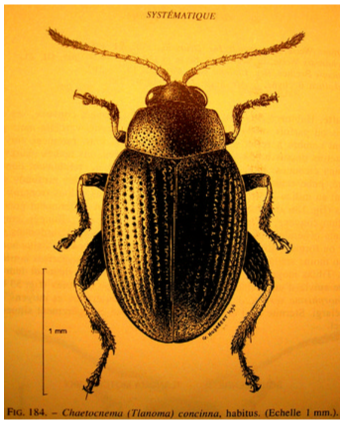
SYSTÉMATIQUE
1 mm
FIG. 184. – Chaetocnema (Tlanoma) concinna, habitus. (Echelle 1 mm.).

Figure 2. Sketch of the Chaetocnema Concinna flea beetle specie.

Overview

Process

The process consisted of the following four steps:

1) Models - Creation of 3D models in OpenGL.
2) Integration – Integration of the models into VRGobi.
3) Interaction - Use of VRJuggler functionality for interaction.
4) Motion - Generation of individual motion patterns for each beetle.

Structure

For modularity the code have been divided into three categories:

1) Code that is used from VRGobi to call the modelling library.
2) The modelling library.
3) The geometry library.

The first category is simply the function calls used in VRGobi to call the modelling library. These consist of four short code segments and are described in more detail in "Integration", below.
The second category, the modelling library, deals with how the beetles are represented in VRGobi. This code takes care of the motion, the interaction and the uniqueness of every beetle.
The third category is the geometry library and consists of OpenGL code that draws a beetle. This code draws a beetle according to the provided parameters. The size, proportions, color, etc is determined by the modelling library that calls this code.

It felt intuitively to separate these three categories by placing them in different subdirectories:

VRGobi

directory:	[VRGobi base directory]
modified files:	wombat.C Makefile

Modelling library

Directory:	[VRGobi base directory]/modelling
files:	modelling.cpp modelling.h definitions.h Makefile

Geometry library

Directory:	[VRGobi base directory]/modelling/beetle
files:	beetle.cpp beetle.h Makefile

Global variables and constants

definitions.h and *beetle.h* contain most of the parameters that one may wish to change. Some parts are hard coded though, such as the motion and the warping of the measurement data. The global variables and constants of interest are listed in Appendix C.

Models

3D Studio

Creating the models directly in OpenGL is not a good approach, considering that a graphical user interface would provide an intuitive way of designing the beetle instead of "programming the looks". Therefore a number of programs were considered for the initial design. 3D Studio (Kinetix) was chosen since it's one of the best 3D modelling programs and there are a lot of plugins/tools available. Modelling the beetle in 3D studio was fairly easy, the model was built up with distorted spheres and cubes, as shown in figure 3.

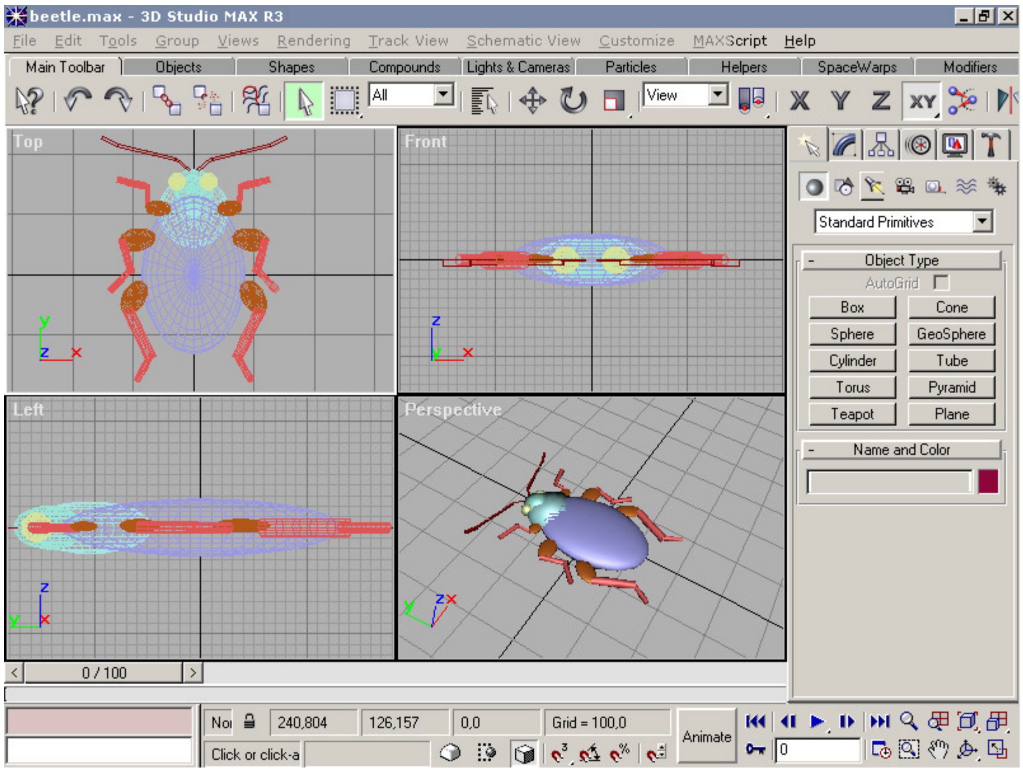

Figure 3. The 3D studio graphical user interface and different views of the modelled beetle.

Polytrans

Polytrans (Okino computer graphics) is a 3D format converter that converts between lots of different 3D formats. It announces that it will transform 3D studio files to OpenGL source code, but that didn't quite work as expected. The generated code from the exported .3ds file did not have the colors, rotations or positions of the different segments. Instead everything was drawn at the origin, in grey. The second approach of converting from an exported .dxf file resulted in a 2 MB C source file, which was highly undesirable.

OpenGL

At this point it seemed like modelling the beetles directly in OpenGL maybe wasn't that bad after all. It wasn't too complicated, since the geometries were rather simple and the 3D studio model was helpful in terms of providing length, rotation and scale of the segments. With a direct representation in OpenGL, greater flexibility and control was achieved over the models. See figure 4.

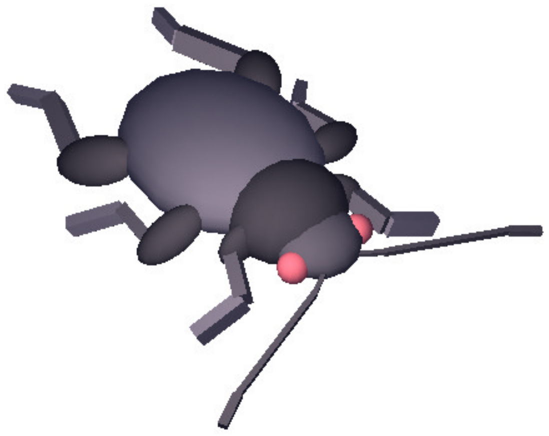

Figure 4. The final OpenGL model of the flea beetle.

The code

The geometry is all placed in beetle.cpp, with definitions in beetle.h. It is fairly straightforward with the following five functions:

```
void drawBeetle(GLint legFactor, GLint headFactor, GLint bodyFactor, GLint angle,
                GLint resolution);
void proportions (GLint *width, GLint *height, GLint *depth);
void findIntersection (GLfloat x1,GLfloat y1,GLfloat z1,GLfloat x2,GLfloat y2,
                       GLfloat z2, GLint zlevel, GLfloat *x, GLfloat *z);
void alexSolid(GLint x, GLint y, GLint z, GLint type, GLint resolution);
void alexJoint(GLint x, GLint y, GLint z, GLint rx, GLint ry, GLint rz, Glint
               type, GLint resolution);
```

drawBeetle, uses *legFactor*, *headFactor* and *bodyFactor* to create an individual geometry when drawing the beetle. These are set by the modelling library and are determined by the six measurements of the beetles. The angle is used to determine where in the motion the beetle is currently. The resolution determines the graphical quality of the beetles.

proportions simply return the default width, height and depth of the model as stated in beetle.h.

findIntersection finds the intersection in the z-plane between two points, which is used to determine when the wand points at a beetle.

alexSolid and *alexJoint* are help functions that are used to simplify *drawBeetle*.

Integration

Due to the proportions of the VRGobi project, a modular approach was chosen. This means that the models can be modified and replaced flexibly, affecting the VRGobi code base minimally. The whole extension is taken care of with four short code segments in the VRGobi code. The extension itself is placed in a

subdirectory called modelling and all its functions have the "modelling_"-prefix for clarity and ease of identification.
The geometry is in turn placed in another subdirectory in order to show that it is possible to have different geometries and switch among them, although using the same modelling library. The abstraction is done at two levels, first to separate VRGobi from the modelling, and second, to separate the modelling from the geometry.

The code

To fully understand what is done at the VRGobi level, it is necessary to take a look at the four code segments in *wombat.C* that make use of the modelling library. As stated earlier, these added lines are the only modifications of the VRGobi source files.

Segment 1

```
//modelling_includes
#include "modelling/modelling.h"
#include "modelling/definitions.h"
```

This segment simply includes *modelling.h* and *definitions.h*. The former consists of the required function prototypes and the latter contains all constants and global variables that are needed (see Appendix C and Appendix D). *definitions.h* is the file where one can experiment with different settings for the library.

Segment 2

```
//modelling_updates
modelling_update(&pathAngle, &rotAngle, angle_inc);

vjMatrix *actualWandMatrix;
jugInter_->getWandMatrix(actualWandMatrix);
modelling_wand(actualWandMatrix, intersection, FLOORLEVEL);
```

This part updates the path- and rotation angles according to a predefined increment. Afterwards, the wand matrix is extracted and the position where the wand intersects with the floor is calculated. The coordinates are set in the *intersection* vector.

Segment 3

```
//modelling_highlightning
modelling_highlighting(i, scale_glyphs, last_selection);
```

Nothing but a call to the highlighting function, which checks if the actual glyph should be highlighted, i.e. checks if this glyph is the currently selected.

Segment 4

```
//modelling_geometry
jugInter_->mButton0_->getData() == vjDigital::ON ? button[0] = 1: button[0] = 0;
jugInter_->handPointing() ? button[1] = 1: button[1] = 0;
//jugInter_->handTwoPointing() ? button[2] = 1: button[2] = 0;
//jugInter_->handClosed()? button[3] = 1: button[3] = 0;
jugInter_->handOpen()? button[4] = 1: button[4] = 0;

vjMatrix *wandMatrix;
jugInter_->getWandMatrix(wandMatrix);
modelling_geometry(raw_data, pnts, X, Y, Z, i,
                   pathAngle, rotAngle, &angle_inc,
                   MAXSPEED, MINSPEED, FREEZE,
                   SPREAD, BSF, PSF, FLOORLEVEL,
                   NO_OF_BEETLES, intersection,
                   enlargement, &selected, &last_selection,
                   wandMatrix, button, resolution);
```

Finally, segment 4 uses VRJuggler to see if the wand buttons are pressed and how. Then it extracts the wand matrix and sends all the information required to the modelling library, which will draw the geometries.

Interaction

It turned out to be fairly straightforward to use the VRJuggler functionality for interaction implementation. The sought information of finding the wand position and whether a button was currently pressed was not a very complicated problem. The interaction consists of the ability to point and click at a beetle to select it. The selected beetle and its corresponding point will be emphasized.

The code

The use of VRJuggler functionality takes place in *wombat.C* at Segment 2 and 4, as defined above. The essential parts are the *vjMatrix* class and the *jugInter_* object. The *vjMatrix* class contains all the matrix functionality needed in this application and *jugInter_* provides access to the current instance of VRJuggler.
Looking in *wombat.h* one see that it is declared as: *stJugglerInterface *jugInter_;*
Looking in *stJugglerInterface.h* one find the functions for VRJuggler access that were used.

Motion

Body motion

The generated motion is not strictly representational but effort has been made to make the motion appear natural. The principle is to let each segment oscillate between two positions according to a sinusoidal function. A hierarchical structure of the beetle is used to manage the motion in a state-like manner. The beetle has a certain value for it's "angle" at each time step, which is propagated through the structure to determine the actual position of the body.

Path motion

The beetles are moving randomly in non-concentric circles with different radius and speed. The model gives the impression of the beetles "running around" in disorder.

The code

The body motion is implemented in *beetle.cpp* in the function that draws the beetle:

```
void drawBeetle(GLint legFactor, GLint headFactor, GLint bodyFactor, GLint angle,
                GLint resolution);
```

All segments in drawBeetle are depending on the *angle* parameter.

The path motion is implemented in the modelling library through two functions in *modelling.cpp*:

```
void modelling_update(GLfloat *pathAngle, GLfloat *rotAngle, GLfloat angle_inc);

void modelling_geometry(float **raw_data, float pnts[][3], int, int, int, int i,
                 GLfloat pathAngle, GLfloat rotAngle, GLfloat *angle_inc,
                 GLint MAXSPEED, GLint MINSPEED, GLint FREEZE,
                 GLfloat SPREAD, GLfloat BSF, GLfloat PSF, GLfloat FLOORLEVEL,
                 GLint NO_OF_BEETLES, GLfloat intersection[],
                 GLint enlargement, GLint *selected, GLint *last_selection,
                 vjMatrix *wandMatrix, GLint button[], GLint resolution);
```

modelling_update updates the *path_angle* and the *rot_angle*, they determine where the beetle is drawn and how the beetle is rotated (so that it always faces the direction of movement).
modelling_geometry does most of the job and would be the equivalent to a main function in the library. Among many tasks that it has, it also generates the beetle specific movement.

Discussion

Due to that it's written by many different people, consists of a large and segmented code base and have a somewhat unclear structure, VRGobi is a project that in my opinion is fairly hard to grasp. Without the assistance of Dianne Cook and Manuel Suarez, the work would've been a lot more troublesome. The challenge of extending the existing program package in a modular way has been a very interesting experience. I believe that the final result has become a flexible and extendable solution to the problem. With just a few modifications in the VRGobi code, all the new functionality is obtained. The libraries are built in such a way that they easily can be replaced and improved thanks to a strong abstraction barrier. A positive side effect is that the libraries are detachable and thus can be reused without modification. They are also platform independent because of the use of OpenGL and can be compiled on any OpenGL supported platform.

Obviously, there are a number of improvements that can be made. The overall code could be "cleaned up", removing some hard coded parts. It would be desirable to separate the motion part from the modelling library, so that one, for instance, could switch between different motions patterns. Similar enhancements could be made with the interaction behaviours and the geometric models. To improve performance, precalculation should be used instead of, as now, everything being calculated in real time.

Another intention was to allow the user to change geometry and interaction properties during runtime. VRGobi already uses most of the input methods, such as key presses and command line arguments, which complicates for a third-party module to make use of them. This is the main reason why it wasn't implemented, although the code is easily extendable for such functionality. That way, the parameters that are currently set and modified in the header files and the source code could be modified in runtime. That would make it a lot easier to experiment with the models and use them more efficiently as part of the visualization tool.

The project has lead to a solution to the initial problem – although there are still some parts that can be improved with regard to greater flexibility. The project has hopefully provided new ideas and issues regarding third-party development for VRGobi and similar projects, apart from showing one way of how to solve the problem.

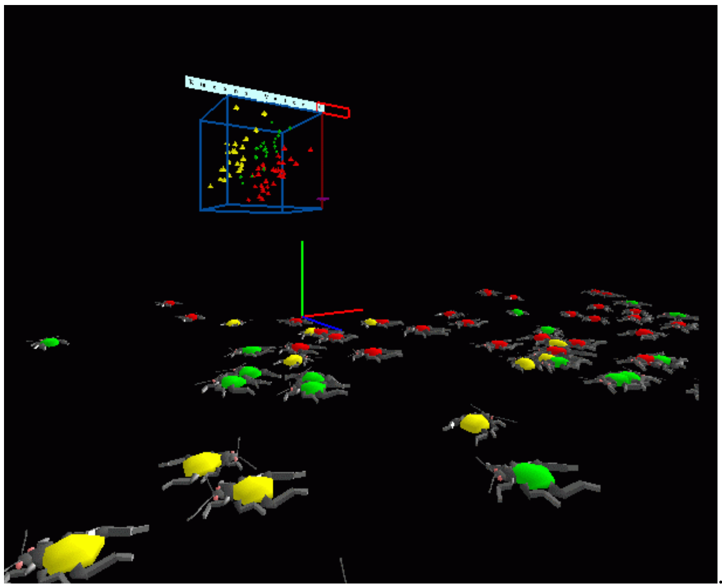.

Figure 5. Screenshot from the application.

References

Angle, E. *Interactive Computer Graphics: A Top-Down Approach with OpenGL*. 1999.
Watt, A. *3D Computer Graphics, Second Edition*. 1999.
Kernigan, B., Ritchie, D. *The C programming language*. 1998.
Prata, S. *C++ programmering (Swedish)*. 1993.
Taxén, G. *En kort introduktion till OpenGL (Swedish)*. 2000.

3D studio max *R3 http://www.kinetix.com*
Polytrans *http://www.okino.com*
OpenGL *http://www.opengl.org*
VRJuggler *http//www.vrjuggler.org*
VRGobi *http://www.public.iastate.edu/~dicook/research/C2/statistic.html*

Appendix A - Dynamic Statistical Data Exploration

Dynamic statistical graphics enables data analysts in all fields to carry out visual investigations leading to insights into relationships in complex data that consists of many different variables. The data consists of multiple observations taken on the same object or measured at the same place. Dynamic statistical graphics involves methods for viewing data in the form of point clouds or modeled surfaces. Higher dimensional data can be projected into 1-, 2- or 3-dimensional planes in a set of multiple views or as a continuous sequence of views that constitutes motion through the higher dimensional space containing the data. There is a strong history of statistical graphics research on developing tools for visualizing relationships between many variables. Much of this work is documented in videos available from the American Statistical Association Statistical Graphics Section Video Lending Library (contact: dfs@research.att.com). In the C2 environment we plan to examine the familiar tools in a new technology and to use the special features of this virtual reality environment to develop completely new tools for the visualization of high dimensional data. The applications of the work will extend to almost all areas of science. In particular we will examine spatially dependent data, for example, data collected over geographical domains for environmental assessment, and agricultural applications.

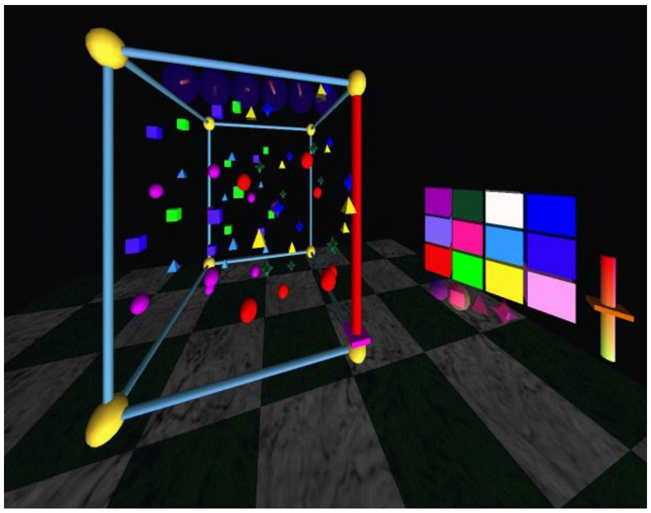

Figure 5. Screenshot from VRGobi.

More information available at *http://www.public.iastate.edu/~dicook/research/C2/statistic.html*

Appendix B - Data set

The data is from a paper by A. A. Lubischew, "On the Use of Discriminant Functions in Taxonomy", Biometrics, Dec 1962, pp.455-477. It contains data on the genus of flea beetle *Chaetocnema*, which contains three species: *concinna*, *heikertingeri*, and *heptapotamica*. The data contains six measurements on each:

1. Width of the first joint of the first tarsus in microns (the sum of measurements for both tarsi).
2. The same for the second joint.
3. Maximal width of the head between the external edges of the eyes in 0.01 mm.
4. Maximal width of the aedeagus in the forepart in microns.
5. Front angle of the aedeagus (1 unit = 7.5 degrees).
6. The aedeagus width from the side in microns.

Appendix C – Constants and global variables

Constants in definitions.h

FLOORLEVEL
Determines the height at which the floor is located.

NO_OF_BEETLES
Limits how many beetles to draw. If *NO_OF_BEETLES* = n > 0, the n first beetles will be drawn.

BSF, PSF
Beetle- and path scale factor, determines the scale of the beetles and the path.

SPREAD
Sets how far apart the beetles are.

FREEZE
If set to 1, the beetle are "frozen", i.e. no motion.

MINSPEED, MAXSPEED
The interval in which the beetle speed can vary.

SPEEDFACTOR
Sets the overall speed.

Constants in beetle.h

```
BODY_WIDTH, BODY_HEIGHT, BODY_DEPTH
Defines the width, height and depth of the beetle.
```

Global variables in definitions.h

intersection[3]
Coordinates for the wand intersection with the floor.

button[5]
Shows if button[0..4] are pressed. Indicates pressed button with a '1' at the corresponding position.

resolution
The resolution of the geometries.

pathAngle, rotAngle
Used to calculate the actual position on the path and the rotation to keep the beetle faced in the direction of movement.

angle_inc
Sets the steps for the increment of the path- and rotAngle.

selected, last_selection
Index of the selected and the previously selected beetle

enlargement
Determines if the selected beetle should be enlarged.

Appendix D – Code

Modelling.h

```
#include "beetle/beetle.h"
#include <vjConfig.h>
#include <Math/vjMatrix.h>
#include <Math/vjVec3.h>

void modelling_geometry(float **raw_data, float pnts[][3], int, int, int, int i,
                 GLfloat pathAngle, GLfloat rotAngle, GLfloat *angle_inc,
                 GLint MAXSPEED, GLint MINSPEED, GLint FREEZE,
                 GLfloat SPREAD, GLfloat BSF, GLfloat PSF, GLfloat FLOORLEVEL,
                 GLint NO_OF_BEETLES, GLfloat intersection[],
                 GLint enlargement, GLint *selected, GLint *last_selection,
                 vjMatrix *wandMatrix, GLint button[], GLint resolution);

void modelling_wand(vjMatrix *actualWandMatrix, GLfloat intersection[], GLfloat
FLOORLEVEL);
void modelling_update(GLfloat *pathAngle, GLfloat *rotAngle, GLfloat angle_inc);
void modelling_highlighting(GLint i, GLfloat scale_glyphs, GLint last_selection);
```

Modelling.cpp

```
#include "modelling.h"

void
modelling_update(GLfloat *pathAngle, GLfloat *rotAngle, GLfloat angle_inc)
{
  *pathAngle += (PI/180)*angle_inc;
  *rotAngle += angle_inc;
  *pathAngle >= 2*PI ? pathAngle = 0:0; //pathAngle %= 360  Cannot use 'cause of
float;
  *rotAngle >= 360 ? rotAngle = 0:0;     //rotAngle %= 360  Cannot use 'cause of
float;
}

void
modelling_wand(vjMatrix *actualWandMatrix,
           GLfloat intersection[], GLfloat FLOORLEVEL)
{
  vjVec3 wp; //Wand Position
  vjVec3 wt; //tip of pointer
  wp.set(0,0,0);
  wt.set(0,0,-10);

  wp.xformFull(*actualWandMatrix, wp);
  wt.xformFull(*actualWandMatrix, wt);

  glColor3f(1,0.5,0);

  glPushMatrix();
    glLoadIdentity();
    glBegin(GL_LINES);
    glVertex3f(wt.vec[0],wt.vec[1],wt.vec[2]);
    glVertex3f(wp.vec[0],wp.vec[1],wp.vec[2]);
    glEnd();
  glPopMatrix();

  GLfloat xIntersection;
  GLfloat zIntersection;

  if (wt.vec[2] < FLOORLEVEL)
    findIntersection(wp.vec[0],wp.vec[1],wp.vec[2], wt.vec[0],
                wt.vec[1],wt.vec[2], FLOORLEVEL,
                &xIntersection, &zIntersection);

  glPushMatrix();
     glLoadIdentity();
```

```
      //Draw the "floor"
      /*      glColor3f(0.5,0.5,0.5);
           glBegin(GL_QUADS);
              glVertex3f(-10,FLOORLEVEL,-10);
            glVertex3f(10,FLOORLEVEL,-10);
            glVertex3f(10,FLOORLEVEL,10);
            glVertex3f(-10,FLOORLEVEL,10);
           glEnd();
      */

      /* //Draws a line from the origin to the intersection point
      glColor3f(1,0,0);
      glLoadIdentity();
      glBegin(GL_LINES);
        glVertex3f(0,0,0);
        glVertex3f(xIntersection,FLOORLEVEL,zIntersection);
      glEnd();
      */

  intersection[0] = xIntersection;
  intersection[1] = FLOORLEVEL;
  intersection[2] = zIntersection;

  glPopMatrix();

}

void
modelling_highlighting(GLint i, GLfloat scale_glyphs, GLint last_selection)
{
  if (i == last_selection)
    {
      glScalef(scale_glyphs * 1.2, scale_glyphs * 1.2, scale_glyphs * 1.2);
      glColor3f(0.8, 0.8, 1);
    }
}

void
modelling_geometry(float **raw_data, float pnts[][3], int X, int Y, int Z, int i,
              GLfloat pathAngle, GLfloat rotAngle, GLfloat *angle_inc,
              GLint MAXSPEED, GLint MINSPEED, GLint FREEZE,
              GLfloat SPREAD, GLfloat BSF, GLfloat PSF, GLfloat FLOORLEVEL,
              GLint NO_OF_BEETLES, GLfloat intersection[],
              GLint enlargement, GLint *selected, GLint *last_selection,
              vjMatrix *wandMatrix, GLint button[], GLint resolution)
{
  //Using the second & fifth measurement to generate an
  //animal-specific path (without caring what they actually represent)
  int a = raw_data[i][2] * PSF;  // x radius of path
  int b = raw_data[i][4] * PSF;  // z radius of path
  int c = raw_data[i][0] * PSF;  // x offset
  int d = raw_data[i][1] * PSF;  // z offset

  a %= 100;
  b %= 100;

  if (a < 50)
    a += 50;

  if (b < 50)
    b += 50;

  /*if (i > 60)
    {
    a = -50;
    b = -50;
    rotAngle = -2*rotAngle;
    pathAngle = -2*pathAngle;

    }*/

  if (i % 2 == 0)
    {
      a *= -1;
      b *= -1;;
```

```
        //          c = i * 5 % 150;
        //d = i * 3 % 300;

        rotAngle *= -1;
        pathAngle *= -1;
      }

    /*
      else if (i % 3 == 0)
      {
      c = i * 2 % 110;
      d = i * 6 % 30;

      rotAngle *= -3;
      pathAngle *= -3;
      }
      else if (i % 5 == 0)
      {
      c = i * 10 % 15;
      d = i * 22 % 200;

      rotAngle *= 2;
      pathAngle *= 2;
      }*/

    float speed;

    speed = (i % (MAXSPEED - MINSPEED))* (*angle_inc) + MINSPEED;
    rotAngle *= speed;
    pathAngle *= speed;

    float animX = a*cos(pathAngle)+c;
    float animZ = b*sin(pathAngle)+d;

    if (FREEZE == 1 )
      {
        animX = 0;
        animZ = 0;
      }

    GLfloat cx = pnts[i][X] + animX;
    GLfloat cy = FLOORLEVEL;
    GLfloat cz = pnts[i][Z] + animZ;

    glPushMatrix();
       glLoadIdentity();
       glTranslatef(pnts[i][X] * SPREAD, FLOORLEVEL, pnts[i][Z] * SPREAD);
       glScalef(PSF, PSF, PSF);
       glTranslatef(animX,0,animZ); //Put the animal on it's path;

       glRotatef(-rotAngle,0,1,0);      // Rotate body to be facing with direction
       glRotatef(90,1,0,0); // Flip down the beetle
       glScalef(BSF,BSF,BSF);

       //Mapping of the statistical data to the beetles
       GLint leg1 = raw_data[i][0];
       GLint leg2 = raw_data[i][1];
       GLint head = raw_data[i][2];
       GLint aedeagus1 = raw_data[i][3];
       GLint aedeagus2 = raw_data[i][4];
       GLint aedeagus3 = raw_data[i][5];

        //Warping of the of the statistical data to the beetles
       GLfloat leg_scale = ((GLfloat)(leg1 + leg2) - 250) / 10;
       GLfloat head_scale = ((GLfloat)(head - 50)) / 5;
       GLfloat body_scale = (GLfloat)(aedeagus1 - 120 + aedeagus2  - 10 + aedeagus3
  - 60) / 20;
       //button[trigger, pointing, twopointing, closed, open]


       //     if (button[0] == 1 && button[2] == 1)
       //  (*angle_inc) -=0.0001;
       // else if (button[0] == 1 && button[3] == 1)
       //  (*angle_inc) +=0.0001;
```

```
        if (i == *last_selection)
          {
         glPushMatrix();
         glScalef(2,2,2);
         glColor3f(0.8, 0.8, 1);
         if (button[4] == 1 && *last_selection != -1)
           enlargement = 1;
         else
           enlargement = 0;

         if (enlargement)
           {
             glPushMatrix();
                cout<<"Enlargement mode"<<endl;
              glLoadIdentity();
              glMultMatrixf(wandMatrix->getFloatPtr());
              glScalef(BSF,BSF,BSF);
              glScalef(PSF,PSF,PSF);
              glTranslatef(-1,0,0);
              glScalef(2,2,2);
              drawBeetle(leg_scale,head_scale,body_scale,0,resolution + 4);
             glPopMatrix();
           }
          }

        if (i > NO_OF_BEETLES || NO_OF_BEETLES < 0)
          drawBeetle(leg_scale,head_scale,body_scale,pathAngle * 1000,resolution);
        //scale of leg, head, body & angle & resolution

        //          cout<<pathAngle<<endl;

        if (i == *last_selection)
          {
         glPopMatrix();
          }

        /* Start of bounding box */
        GLfloat w,h,d2;
        GLint ab, cd, ef;
        proportions(&ab,&cd,&ef);
        w = (GLfloat)ab * PSF * BSF;
        h = (GLfloat)cd * PSF * BSF;
        d2 = (GLfloat)ef * PSF * BSF;

        animX *= PSF;
        animZ *= PSF;
        //


        cx = animX + (float)pnts[i][X] * SPREAD; //the world coordinate of the
beetle
        cz = animZ + (float)pnts[i][Z] * SPREAD;

        if (button[4] == 1)
          cout<<"open"<<endl;
        else if (button[0] == 1)
          cout<<"1"<<endl;

        if (button[0] == 1)
          {
         GLfloat xi = intersection[0];
         GLfloat zi = intersection[2];

         if (xi > cx - w && xi < cx + w &&
             zi > cz - h && zi < cz + h)
           {
             *last_selection = i;
             *selected = i;

             glScalef(2,2,2);
             glColor4f(1,1,1,1);
             glBegin(GL_QUADS);
             glVertex3f(w,h,d2);
             glVertex3f(-w,h,d2);
             glVertex3f(-w,-h,d2);
```

```
              glVertex3f(w,-h,d2);
              glVertex3f(w,h,d2);
              glEnd();

              cout<<"Beetle "<<*selected<<" last_selection"<<endl;
            }
           }

     glPopMatrix();
   }
```

Definitions.h

```
#define FLOORLEVEL   0 //-1.5
#define NO_OF_BEETLES -1 //actually how many beetles to skip, -1 = all
#define BSF 0.01 //0.01 //beetlescalefactor
#define PSF 0.1 //0.01 //pathscalefactor
#define SPREAD 15
#define FREEZE 0
GLfloat SPEEDFACTOR = 0.05;
GLint MAXSPEED = 10;
GLint MINSPEED = 3;

GLint resolution = 3;

GLint angle = 0;
GLfloat pathAngle = 0;
GLfloat rotAngle = 0;
GLint turn = 0;
GLfloat angle_inc = SPEEDFACTOR;
GLint last_selection = -1;
GLint enlargement = 0;
GLint counter = 0;
GLint selected = -1;
GLfloat intersection[3] = {0,0,0};
GLint button[5] = {0,0,0,0,0};
```

Beetle.h

```
#ifdef WIN32
#include <windows.h>
#endif

#include <GL/glut.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define SPHERE 1
#define CUBE 2
#define PI 3.14 //3.141592654
#define PI_180 0.1745

#define BODY_WIDTH 160
#define BODY_HEIGHT 210
#define BODY_DEPTH 80


void drawBeetle(GLint legFactor, GLint headFactor, GLint bodyFactor, GLint angle,
GLint resolution);
void proportions (GLint *width, GLint *height, GLint *depth);
void findIntersection (GLfloat x1,GLfloat y1,GLfloat z1,GLfloat x2,GLfloat
y2,GLfloat z2, GLint zlevel, GLfloat *x, GLfloat *z);
void alexSolid(GLint x, GLint y, GLint z, GLint type, GLint resolution);
void alexJoint(GLint x, GLint y, GLint z, GLint rx, GLint ry, GLint rz, GLint
type, GLint resolution);
//The resolution is only used for spheres
```

Beetle.cpp

```
#include "beetle.h"

void
drawBeetle(GLint legFactor, GLint headFactor, GLint bodyFactor, GLint angle,
GLint resolution)
{
  GLfloat clj1 = 0.3; // Color leg joint 1
  GLfloat clj2 = 0.4;
  GLfloat clj3 = 0.5;
  GLfloat ca = 0.35; // Color antenn
  GLfloat cf = 0.4; // Color forehead
  GLfloat ch = 0.3; // Color head
  GLfloat ce[] = {1, 0.5, 0.5};

  GLint vla = 20; // Vertical leg angle

  angle *= PI_180;
  GLfloat sin_angle = sin(angle);
  GLfloat cos_angle = cos(angle);

  GLint laa1 = 60 + 3*sin_angle; //left antenna angle 1st joint
  GLint laa2 = 10 + 2*sin_angle;
  GLint laa3 = -10 + 6*sin_angle;
  GLint raa1 = 60 + 3*sin(angle - PI_180 * 4);
  GLint raa2 = 10 + 2*sin(angle - PI_180 * 4);
  GLint raa3 = -10 + 6*sin(angle - PI_180 * 4);

  GLint ll1j1a = 15*cos_angle; //left leg 1 joint 1 angle
  GLint ll2j1a = 10*sin(angle - PI_180 * 10);
  GLint ll3j1a = 12*sin(angle - PI_180 * 15);
  GLint rl1j1a = 15*cos(angle - PI_180 * 30);
  GLint rl2j1a = 10*sin(angle - PI_180 * 40);
  GLint rl3j1a = 12*sin(angle - PI_180 * 45);

  GLfloat hva = 3 * cos_angle; // head vertical angle - bounce
  GLfloat hha = 3 * sin_angle; // head horizotal angle - lookaround

  GLint al1 = 70; //antenna length, 1st segment
  GLint al2 = 200;
  GLint al3 = 50;

  GLint bx = BODY_WIDTH + bodyFactor;
  GLint by = BODY_HEIGHT + bodyFactor;
  GLint bz = BODY_DEPTH + bodyFactor;
  GLint hx = 100 + headFactor;
  GLint hy = 90 + headFactor;
  GLint hz = 70 + headFactor;
  GLint fy = 70;
  GLint fx = 70;
  GLint fz = 40;
  GLfloat l = 1 + legFactor*0.1;

  // Bounding box will be regarding the body.
  GLfloat top = by;
  GLfloat right =  bx;
  GLfloat left =  -right;
  GLfloat bottom =  -by;
  GLfloat thickness = bz;

  /*  glPushMatrix();
     glScalef(2*bx, 2*by, bz);
     glutWireCube(1);
  glPopMatrix();
  */

  glPushMatrix();
     //glTranslatef(0,15,0);

     alexSolid(bx,by,bz,SPHERE,resolution + 2); //body
     //(the body will be coloured with the actual opengl color

     glTranslatef(0,by * 0.9,0);


     glPushMatrix();
    glColor3f(ch,ch,ch);
    glRotatef(hva,1,0,0);
    glRotatef(hha,0,1,0);
        alexSolid(hx,hy,hz,SPHERE,resolution); // head
    glTranslatef(0,0.8*hy,0);
    glColor3f(cf,cf,cf);
    alexSolid(fx,fy,fz,SPHERE,resolution); // forehead

    glPushMatrix(); // draw the eyes
           glTranslatef(fy*0.9,hy*0.23,0);
        glColor3f(ce[0],ce[1],ce[2]);
        alexSolid(25,25,25,SPHERE,resolution); // right eye
        glTranslatef(-fy*0.9*2,0,0);
        alexSolid(25,25,25,SPHERE,resolution); // left eye
    glPopMatrix();

    glTranslatef(0,fy*0.6,0); //  antenna position in forehead

    glColor3f(ca,ca,ca);
    //draw antennas
    int i;

    for (i = 1; i > -2; i -= 2)
    {
      GLint aa1 = raa1;
      GLint aa2 = raa2;
      GLint aa3 = raa3;

      if (i > 0)
        {
           aa1 = laa1;
           aa2 = laa2;
           aa3 = laa3;
        }


        glPushMatrix();
           glTranslatef(i*-fx*0.1,0,0);
           alexJoint(7,al1,7,0,0,i*aa1,CUBE,resolution);
```

```
             alexJoint(9,al2,10,0,0,i*aa2,CUBE,resolution);
             alexJoint(13,al3,11,0,0,i*aa3,CUBE,resolution);
          glPopMatrix();
      }
     glPopMatrix(); // back at center of head
     glPushMatrix();
         GLint legResolution = resolution;
      //draw legs
      for (i = 1; i > -2; i -= 2)
        {
          GLint a1 = rl1j1a;
          GLint a2 = rl2j1a;
          GLint a3 = rl3j1a;

          if (i > 0)
            {
            a1 = ll1j1a;
            a2 = ll2j1a;
            a3 = ll3j1a;
            }

          glPushMatrix(); // Leg 1
                glTranslatef(i*-bx*0.45,0,0);
                glRotatef(i*vla,0,1,0);
             glScalef(l,l,l);
             glColor3f(clj1,clj1,clj1);
             alexJoint(20,40,30,0,0,i*70 + a1,SPHERE,legResolution);
             glColor3f(clj2,clj2,clj2);
             alexJoint(20,80,30,0,0,i*-60,CUBE,legResolution);
             glColor3f(clj3,clj3,clj3);
             alexJoint(20,80,30,0,0,i*70,CUBE,legResolution);
             glPopMatrix();

          glPushMatrix(); // Leg 2
                glTranslatef(i*-bx*0.86,-by*0.38,0);
                glRotatef(i*vla,0,1,0);
             glScalef(l,l,l);
             glColor3f(clj1,clj1,clj1);
             alexJoint(30,60,30,0,0,i*100 + a2,SPHERE,legResolution);
             glColor3f(clj2,clj2,clj2);
             alexJoint(20,90,30,0,0,i*90,CUBE,legResolution);
             glColor3f(clj3,clj3,clj3);
             alexJoint(20,90,30,0,0,i*-60,CUBE,legResolution);
             glPopMatrix();

          glPushMatrix(); // Leg 3
             glTranslatef(i*-bx*0.95,-bx*1.7,0);
                glRotatef(i*vla,0,1,0);
             glScalef(l,l,l);
             glColor3f(clj1,clj1,clj1);
             alexJoint(40,70,40,0,0,i*140 + a3,SPHERE,legResolution);
             glColor3f(clj2,clj2,clj2);
             alexJoint(30,150,40,0,0,i*60,CUBE,legResolution);
             glColor3f(clj3,clj3,clj3);
             alexJoint(20,90,30,0,0,i*-40,CUBE,legResolution);
             glPopMatrix();

      }
     glPopMatrix();

   glPopMatrix();
 }

 void
 proportions (GLint *width, GLint *height, GLint *depth)
 {
   *width = BODY_WIDTH;
   *height = BODY_HEIGHT;
   *depth = BODY_DEPTH;

 }

 void alexJoint(GLint x, GLint y, GLint z, GLint rx, GLint ry, GLint rz, GLint
 type, GLint resolution)
 {
   GLfloat offset = (GLfloat)y/2;

   glRotatef(rx, 1, 0, 0);      // rotate around the joint
   glRotatef(ry, 0, 1, 0);
   glRotatef(rz, 0, 0, 1);

   glTranslatef(0, offset, 0);  // move so that we draw at previous joint
   alexSolid(x,y,z,type, resolution);

   glTranslatef(0, offset, 0);  // ends up at the end of this joint
 }

 void alexSolid(GLint x, GLint y, GLint z, GLint type, GLint resolution)
 {
   glPushMatrix();
      glScalef(x,y,z);
      if (type != SPHERE)
        glutSolidCube(1);
      else
        glutSolidSphere(1,resolution,resolution);
   glPopMatrix();
 }

 void
 findIntersection (GLfloat x1,GLfloat y1,GLfloat z1,GLfloat x2,GLfloat y2,GLfloat
 z2, GLint zlevel, GLfloat *x, GLfloat *z)
 {
   //y = y2*u + y1*(1-u) = zlevel = u*(y2-y1) + y1
   //u = (zlevel - y1) / (y2 - y1)

   GLfloat u = (zlevel - y1) / (y2 - y1);
   *x = x2*u + x1*(1-u);
   *z = z2*u + z1*(1-u);

   /*  glPushMatrix();
     glTranslatef(x, zlevel, z);*/
     //cout << x << " " << zlevel << " " << z << endl;
   //glutSolidCube(1);
   //glPopMatrix();
 }
```