SURFAN - Surface Analyzer
DRAFT VERSION

Greger Cronquist   Peter Hedin   Konstantin Karagatsos
Johan Lindberg

University of South Carolina, Columbia, SC, 1997
Last update July 25, 1997

Contents

List of Figures

Part I

Theory

Part II

User's Guide

Chapter 1

General

The purpose of Surfan is to create an optimal triangulation of a surface given by ordinary CAD models. For this purpose, the program creates a graph on the CAD surface, where the nodes represent patches and where edges between two nodes indicate that the corresponding patches are adjoined in the model.

After reading a CAD model from a file, e.g. in the IGES format, it is displayed and the user is able to view it from different angles, calculate the graph (with the possibility to alter it manually if necessary) and then create the desired triangulation. Figure 1.1 shows the overview of Surfan.

In the current program version the program only supports IGES files; the other formats (VDA, STEP etc.) are not yet implemented. The image problem is not yet addressed, and the graph algorithms that will be used are not yet fully developed, and are not implemented. The triangulation window is created but no triangulation exists.

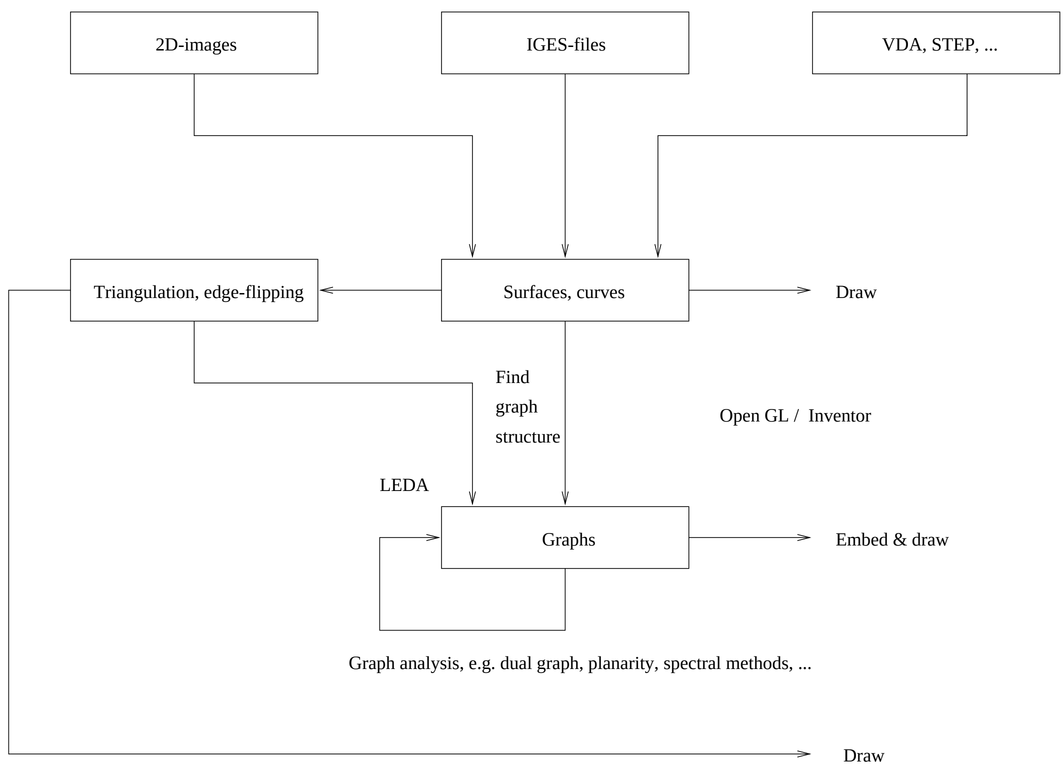

Figure 1.1: Program overview

Chapter 2

User Interface

2.1 Overview

The system consists of three major parts: the Main Window that contains the main menu and from which the other windows are started, the Model and Graph Window in which the user may view a given CAD model to calculate and alter its adjacency graph, and the Triangulation Window for displaying the resulting triangulation of a model.

2.2 Main Window

The main window (figure 2.1) contains the main menu, with the following items:

File - Load IGES Read a CAD model from a file, using the IGES format.

File - Load Graph Read a graph from a file, using the GML [16] format.

File - Save Graph Save a graph to a file, using the GML [16] format.

File - Quit Quit the application.

Window - Model and Graph Enter the Model and Graph Window, where the CAD model may be viewed from different directions and its corresponding graph may be viewed and edited.

Window - Triangulation Enter the Triangulation Window.

Help - About Give information about Surfan.

2.3 Model and Graph Window

In this window (figure 2.2) the CAD model is being displayed. The model is centered around the origin of a coordinate system. The mouse can be used to manipulate the model in several ways.

First you can change the view of the object. When the left mouse button is pressed, moving the mouse causes the object to rotate around its center, making it possible for the user to view it from all sides. Pressing the middle mouse button, moving the mouse pans the view in the direction of movement (hold down the 'Ctrl' key for fast panning). Moving the mouse with the right button pressed causes zooming; a movement up/right zooms in on the object while a down/left motion zooms away from it (hold down the 'Ctrl' key for fast zooming). Pressing the "Space" button returns the viewer to its original position.

Secondly, you can change the graph configuration. Use the 'Ctrl' key together with the left mouse button to select or deselect a node. Select two nodes to add or delete an edge between them. Use the 'Shift' key together with the left mouse button to increase the size of all the nodes. Use the 'Shift' key together with the right mouse button to decrease the size of all the nodes.

The menu in this window contains the following items:

File - Load IGES Read a CAD model from a file, using the IGES format.

File - Load Graph Read a graph from a file, using the GML [16] format.

File - Save Graph Save a graph to a file, using the GML [16] format.

File - Close Close the Model and Graph Window.

View - Filled Polygons View the model with solid surfaces.

View - Filled With Colors View the model with solid surfaces, using one color for each patch, thereby making it easy to see the patches that make up the model.

View - Transparent Polygons View the model using transparent polygons, thereby making it easy to see the nodes and edges through the surfaces.

View - Wireframe View the model as a wireframe.

View - Large Coord Turn the large coordinate system on or off.

View - Small Coord Turn the small, fixed coordinate system on or off.

Graph - View Turn the graph on or off.

Help - Help Gives information about how to navigate in the Model and Graph Window.

2.4 Triangulation Window

In this window the triangulation of a surface is shown. The user is allowed to move around to view the triangulation from different positions, see section 2.3.

File - Close Close the Triangulation Window.

View - Large Coord Turn the large coordinate system on or off.

View - Small Coord Turn the small, fixed coordinate system on or off.

Help - Help Gives information about how to navigate in the Triangulation Window.

Figure 2.1: SurFan Main Window

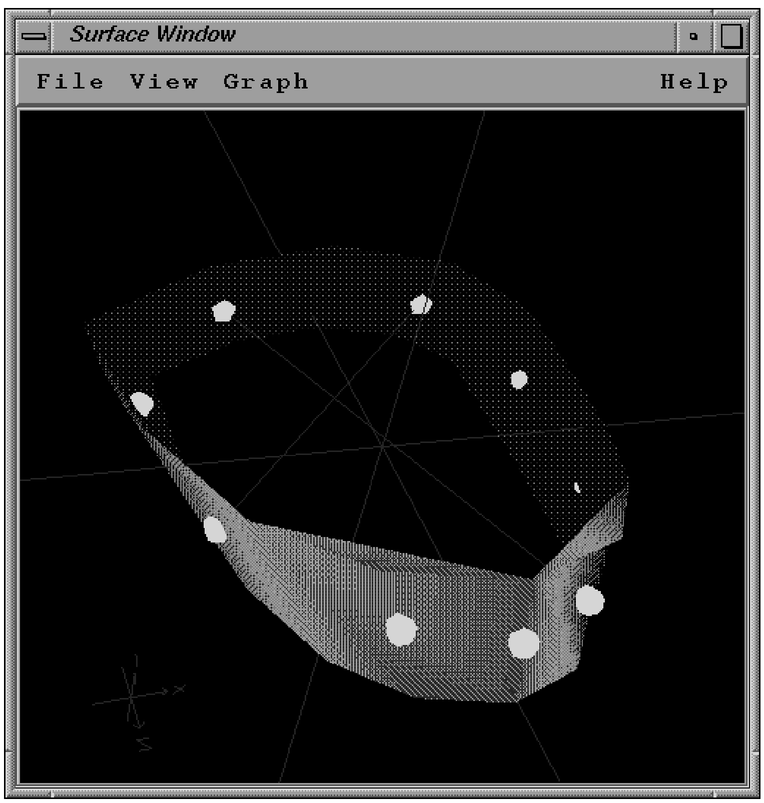

Figure 2.2: Model and Graph Window

Figure 2.3: Triangulation Window

Part III

Implementation

Chapter 3

Overview

3.1 Packages

We are using C++ as an overall programming language, LEDA (Library of Efficient Data types and Algorithms) [18] for the graphs and graph operations, DT_NURBS (David Taylor Non-Uniform Rational B-Splines) [25] for operations concerning calculations on NURBS, [22] Motif [15] for the user interface and OpenGL [21] for visualisation of the models and graphs.

Chapter 4

Class implementation

In this chapter all classes and their relationships are described.

Overview Here the class is described in a general manner; its main purpose is explained. A list of public and protected functions are also provided here.

Member functions Here all public and protected functions are explained.

Dependance on other classes If a class contains an object of some other class, or inherits another class, this is described here.

Examples Here are examples of how to use the class.

In figure 4.1, the hierarchy of classes is described. If a class contains an object of some other class, this is indicated with an arrow to the class in question. If a class is inherited from some other class, this is indicated by placing the two classes within a common frame, with the superclass located above and slightly to the left of the inherited class.

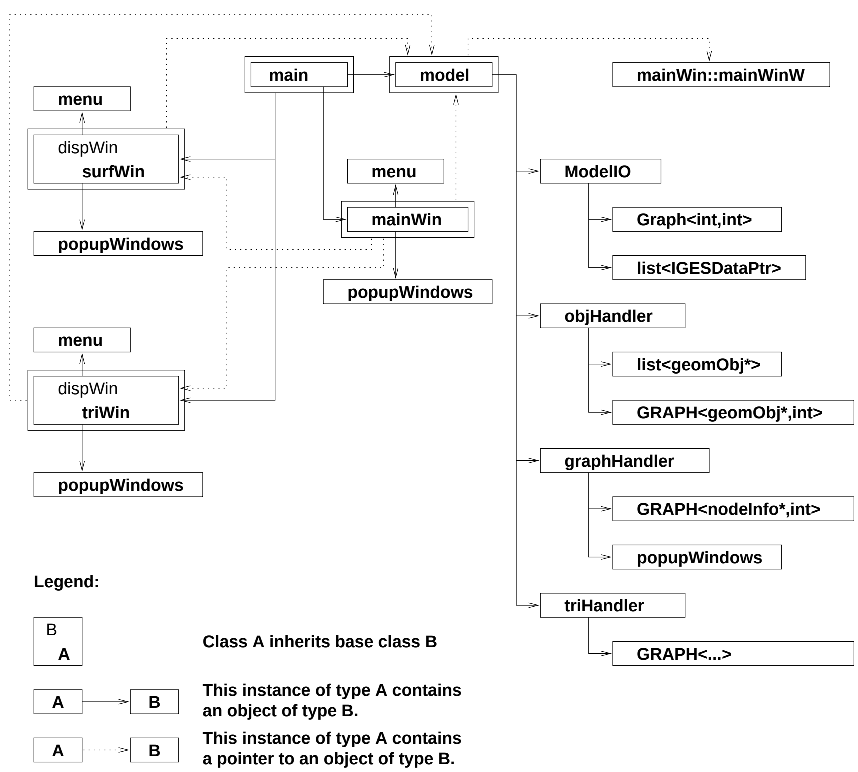

Figure 4.1: Class overview

4.1 baseWin

Overview

The class baseWin implements a popup window with a menu. Start by creating an object of this type, then make a widget that is to be wrapped. This widgets parent is given by getWidget(). When this widget is created, call wrapWidget to put it into a popup window with a menu on top.

baseWin
baseWin(Widget,char *,int,int,int,int,int) void wrapWidget(Widget) Widget getWidget() void popup() void popdown() void addMenuPane(char *) void addHelpMenuPane(char *) void addMenuItem(char *,XtCallbackProc,XtPointer) void addSeparator()

Member functions

baseWin(Widget shell,char *shellName,int n,int x,int y,int w,int h) The constructor makes the popup window. shell is the widget that should act as a parent to this window, shellName is its name, n is the number of menu panes that the menu should contain and the integers x, y, w and h set the position and dimensions of the window. These four last parameters may be omitted, in which case x and y are set to zero, while w and h are set to 100.

void wrapWidget(Widget) After creating a widget that should be "wrapped" by a window, call this function to make it a part of the window.

void popup() This causes the created window to pop up.

void popup() This causes the created window to pop down.

void addMenuPane(char *) This adds a menu pane to the window menu. For details, see 4.22.

void addHelpMenuPane(char *) This adds a menu help pane to the window menu. For details, see 4.22.

void addMenuItem(char *,XtCallbackProc,XtPointer) This adds a menu item to the last menu pane of the window menu. For details, see 4.22.

void addSeparator() This function adds a separator in the current pane.

Widget getWidget() This function returns the form widget that should act as a parent to the widget that will be wrapped by this class.

Dependance on other classes

Uses an object of type menu, see 4.22.

Example

```
#include <iostream.h>    // For the stream output
#include <Xm/Label.h>    // For the label in the popup window
#include <Xm/PushB.h>    // For the button in the main window
#include "basewin.h"     // Include the menu class

void quitCB(Widget w,XtPointer clientData,XtPointer callData){
  exit(0);
}

void choiceOneCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 1" << endl;
}

void choiceTwoCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 2" << endl;
}

void choiceThreeCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 3" << endl;
}

void pushCB(Widget w,XtPointer clientData,XtPointer callData){
  baseWin *bW = (baseWin *) clientData;
  bW -> popup();
}

void setUpTheMenus(baseWin *bW){
  bW->addMenuPane("OneTwo");
  bW->addMenuItem("Choice 1",choiceOneCB,NULL);
  bW->addMenuItem("Choice 2",choiceTwoCB,NULL);
  bW->addSeparator();
  bW->addMenuItem("Quit",quitCB,NULL);
  bW->addHelpMenuPane("Three");
  bW->addMenuItem("Choice 3",choiceThreeCB,NULL);
}
```

```
void main(int argc,char **argv){
  Widget toplevel, label, button;
  XtAppContext app;
  baseWin *bW;

  toplevel = XtAppInitialize(&app,"Test",NULL,0,&argc,argv,NULL,NULL,0);

  /***** Create the popup window *****/

  bW = new baseWin (toplevel,"Test Window",2, 100, 100, 500, 200);
  label = XmCreateLabel(bW->getWidget(), "A label",NULL,0);
  bW->wrapWidget(label);

  /***** Create a "Popup" button in the main window *****/

  button = XmCreatePushButton(toplevel,"Push me!",NULL,0);
  XtManageChild(button);
  XtAddCallback(button, XmNactivateCallback, pushCB, (XtPointer) bW);

  /***** Set up the menus in the popup window *****/

  setUpTheMenus(bW);

  /***** Run the application *****/

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app);
}
```

4.2 color

Overview

This class contains all information needed to draw the model using a different color for each patch.

color
color() void setColor()

Member functions

color() The constructor initializes some internal variables.

void setColor() Inserts a "glMaterialfv(GL_FRONT, GL_DIFFUSE, col)" into the code, i.e. sets the material's diffuse color to col. This color is always another than it was at the last call to this function.

Dependance on other classes

None.

Example

See the source code for "Surfan".

4.3 dispWin

Overview

The class dispWin creates an OpenGL drawing area contained in a scrollable window, with a menu on top.

dispWin
protected Widget canvas protected GLXContext cx protected Display *dpy protected Dimension width protected Dimension height
dispWin(Widget,char *,int,int,int,int,int) void popup() void popdown() void addMenuPane(char *) void addHelpMenuPane(char *) void addMenuItem(char *,XtCallbackProc,XtPointer) void addSeparator() int isActive() virtual void draw(Widget) = 0 virtual void resize(Widget) = 0 virtual void keyPressed(Widget,XEvent *) virtual void getMouse(Widget,XEvent *) virtual void trackMouse(Widget,XEvent *) virtual void releaseMouse(Widget,XEvent *)

Member functions

dispWin(Widget,char *,int,int,int,int,int) The constructor makes the popup window. shell is the widget that should act as a parent to this window, shellName is its name, n is the number of menu panes that the menu should contain and the integers x, y, w and h set the position and dimensions of the window. These four last parameters may be omitted, in which case x and y are set to zero, while w and h are set to 100.

void popup() This causes the created window to pop up.

void popdown() This causes the created window to pop down.

void addMenuPane(char *) This adds a menu pane to the window menu. For details, see 4.22.

void addHelpMenuPane(char *) This adds a menu help pane to the window menu. For details, see 4.22.

void addMenuItem(char *,XtCallbackProc,XtPointer) This adds a menu item to the last menu pane of the window menu. For details, see 4.22.

void addSeparator() This function adds a separator in the current pane.

virtual void draw(Widget) = 0 This function must be written for the derived instance of dispWin. In addition to the given widget, which is the OpenGL drawing area widget, the programmer also has access to the protected members width and height that give the dimensions of the drawing area.

virtual void resize(Widget) = 0 This function must be written for the derived instance of dispWin. This is used to set up new coordinate system, viewing positions etc. The members width and height are set automatically, and don't have to be set by this function.

virtual void keyPressed(Widget,XEvent *ev) This function is called when a key is pressed in the drawing area. ev→xkey.keycode gives information about which key was pressed.

virtual void getMouse(Widget,XEvent *ev) This function is called when a mouse button is pressed in the drawing area. The coordinates of the mouse are given by ev→button.x and ev→button.y respectively. ev→button.state gives information about which button that was pressed (ev→button.state—Button1Mask etc).

virtual void trackMouse(Widget,XEvent *ev) This function is called when the mouse is moved in the drawing area. The coordinates of the mouse are given by ev→button.x and ev→button.y respectively.

virtual void releaseMouse(Widget,XEvent *ev) This function is called when the mouse is moved out of the drawing area. The coordinates of the mouse are given by ev→button.x and ev→button.y respectively.

Dependance on other classes

This class uses an object of type menu, see 4.22.

Example

In this example a new class, derivedDispWin is derived from dispWin, defining its own draw, resize, getMouse, trackMouse and releaseMouse functions as well as functions for creating the menus in the example and for determining which of the mouse buttons was pressed.

```
#include <Xm/PushB.h>
#include "../dispwin.h"

// ***** Callbacks - global for simplicity *****

void quitCB(Widget w,XtPointer clientData,XtPointer callData){
  exit(0);
}

void choiceOneCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 1" << endl;
}

void choiceTwoCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 2" << endl;
}

void choiceThreeCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 3" << endl;
}

// ***** derivedDispWin - the class derived from dispWin *****

class derivedDispWin : public dispWin{
public:
  derivedDispWin(Widget shell,String name,int menuPanes,
                 int x,int y,int w,int h):
    dispWin(shell,name,menuPanes,x,y,w,h) {}
  void setupMenu();
  int determineButton(int);
  void draw(Widget);
  void resize(Widget);
  void getMouse(Widget,XEvent *);
  void trackMouse(Widget,XEvent *);
  void releaseMouse(Widget,XEvent *);
};

void derivedDispWin::setupMenu(){
  addMenuPane("OneTwo");
  addMenuItem("Choice 1",choiceOneCB,NULL);
  addMenuItem("Choice 2",choiceTwoCB,NULL);
  addSeparator();
  addMenuItem("Quit",quitCB,NULL);
  addHelpMenuPane("Three");
  addMenuItem("Choice 3",choiceThreeCB,NULL);
}
```

```
void derivedDispWin::draw(Widget w){
  glXMakeCurrent(dpy, XtWindow(canvas), cx);
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);
  glBegin(GL_LINES);
    glVertex3i(0, 0, 0);
    glVertex3i((int) width, (int) height, 0);
    glVertex3i(0, (int) height, 0);
    glVertex3i((int) width, 0, 0);
  glEnd();
  glFinish();
  glXSwapBuffers(dpy, XtWindow(w));
}

void derivedDispWin::resize(Widget w){
  glViewport (0, 0, (GLsizei) width, (GLsizei) height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, (GLdouble) width, 0.0, (GLdouble) height);
}

int derivedDispWin::determineButton(int button){
  if ((button & Button1) && (button & Button2)
      && (button & Button3) && (button & Button5))
    return 3;
  if ((button & Button2) && (button & Button3))
    return 2;
  if ((button & Button1) && (button & Button3) && (button & Button5))
    return 1;
  return 0;
}

void derivedDispWin::getMouse(Widget w,XEvent *ev){
  cout << "Get mouse: (" << ev->xbutton.x << ","
    << ev->xbutton.y << ")";
  switch(determineButton(ev->xbutton.button)){
    case 1: cout << "Left"; break;
    case 2: cout << "Middle"; break;
    case 3: cout << "Right"; break;
  }
  cout << endl;
}
```

```
void derivedDispWin::trackMouse(Widget w,XEvent *ev){
  cout << "Track mouse: (" << ev->xbutton.x << ","
    << ev->xbutton.y << ")" << endl;
}

void derivedDispWin::releaseMouse(Widget w,XEvent *ev){
  cout << "Clear mouse: (" << ev->xbutton.x << ","
    << ev->xbutton.y << ")" << endl;
}

// ***** Callback to pop up the display window *****

void pushCB(Widget w,XtPointer clientData,XtPointer callData){
  derivedDispWin *dW = (derivedDispWin *) clientData;
  dW -> popup();
}

// ***** The main loop *****

void main(int argc,char **argv){
  Widget toplevel, button;
  XtAppContext app;
  derivedDispWin *dW;

  toplevel = XtAppInitialize(&app,"Test",NULL,0,&argc,argv,NULL,NULL,0);

  /***** Create the display window. *****/
  dW = new derivedDispWin (toplevel,"Test Window",2, 100, 100, 500, 200);
  dW -> setupMenu();

  /***** Create the "Popup" button in the main window *****/
  button = XmCreatePushButton(toplevel,"Popup",NULL,0);
  XtManageChild(button);
  XtAddCallback(button, XmNactivateCallback, pushCB, (XtPointer) dW);

  /***** Realize the toplevel shell *****/
  XtRealizeWidget(toplevel);

  /***** Run the application *****/
  XtAppMainLoop(app);
}
```

4.4 geomObj [0]

Overview

Base-class for all the different object-types that are supported. The methods and variables that are common in all derived classes will only be mentioned here.

0 geomObj
geomObj *parent trMatrix transMatrix int Type void clearTransMatrix() trMatrixPtr matMul(trMatrix a, trMatrix b) vectorPtr vecMul(trMatrix a, vectorPtr v)
geomObj() virtual geomObj() void setParent(geomObj *parent) geomObj *getParent() int getType() trMatrixPtr getTransMatrix() trMatrixPtr getTotalMatrix() void setTransMatrix(trMatrix matrix) virtual draw() virtual done() virtual double getSize() virtual vectorPtr getCenter()

Member functions

geomObj() The constructor only does some intializations.

virtual ~geomObj() The destructor only does som deinitializations.

void setParent(geomObj *parent) Sets the parent of this object to be parent.

geomObj *getParent() Returns the parent of the object or NULL if there is none (which would mean that the object is the root-object).

int getType() Ask object what object-type it contains. geomObj has type 0, the derived classes has types more or less following the IGES standard.

trMatrixPtr getTransMatrix() Returns the object's transformation matrix.

trMatrixPtr getTotalMatrix() Returns the result of all the transformation matrices of the parents of this object multiplied together.

void setTransMatrix(trMatrix *matrix) Set the object's transformation matrix.

virtual draw(bool changeColor, color *col) Method to draw the object. Puts drawing commands on the OpenGL stack. changeColor tells the object that it should change it color using col-¿setColor().

virtual done() Has to be called after all data of an object has been set, does some pre-drawing initializations.

virtual vectorPtr getCenter() Returns a pointer to the coordinate of the approximate center of the object.

virtual double getSize() Returns the approximate size of the object.

Dependence on other classes

geomObj uses itself as well as OpenGL functions and the color class.

Example

See the source code for "Surfan".

4.5 geomObjTransMatrix [0]

Overview

A child of geomObj. This object describes a transfmation matrix. The addition to its parent class is that it has knowledge of its children.

0 geomObjTransMatrix
list¡geomObj *¿ children
list_item pos
void addChild(geomObj *child)
geomObj *getFirstChild()
geomObj *getNextChild()

Member functions

void addChild(geomObj *child) Appends child to children.

geomObj *getFirstChild() Returns a pointer to the first geomObj in children.

geomObj *getNextChild() Returns a pointer to the next geomObj in children, or NULL if there are none, or if you already were at the last child.

Dependence on other classes

geomObjTransMatrix is a child of geomObj that also uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.6 geomObjCircularArc [100]

Overview

A child of geomObj. This objecttype describes a circle that can have an arbitrarily large opening. The object type is 100.

100 geomObjCircularArc
double radius vector start double openAngle
geomObjCircularArc() void setRadius(double radius) void setStart(vector start) void setOpenAngle(double v) double getRadius() vector getStart() double getOpenAngle()

Member functions

geomObjCircularArc() The constructor just makes sure the characteristics are set to zero.

void setRadius(double radius) Simply sets the radius of the circle.

void setStart(vector start) Set the direction in which the left (most anti-clockwise) side of the opening is. If openAngle is zero, start can be an arbitrary vector.

void setOpenAngle(double v) Set how many radians large the opening is. If this is zero we have a full circle.

double getRadius() Just get the radius of the circle.

vector getStart() In which direction does the opening begin?

double getOpenAngle() How large, in radians, is the opening?

Dependence on other classes

geomObjCircularArc is a child of geomObj and uses OpenGL functions that also uses the color class.

Example

See the source code for "Surfan".

4.7 geomObjCompositeCurve [102]

Overview

A child of geomObj. This object-type describes a curve that is composed of several different curve-elements. The object type is 102.

102 geomObjCompositeCurve
list<geomObj *> members list_item currentPos addMember(geomObj *newMember)
geomObjCompositeCurve(int n) ~geomObjCompositeCurve() int getNoOfMembers() void setMember(int n, geomObj *newMember) geomObj *getMember(int n)

Member functions

geomObjCompositeCurve(int n) The constructor initializes all relevant variables and lists. n specifies the number of curves it is made of.

~geomObjCompositeCurve() The destructor frees all allocated memory.

int getNoOfMembers() Returns the number of elements this curve is made up of.

void setMember(int n, geomObj *newMember) Set curve n to the geomObj specified by newMember.

geomObj *getMember(int n)

Dependence on other classes

geomObjCompositeCurve is a child of geomObj that uses LEDA's list class, OpenGL and the color class.

Example

See the source code for "Surfan".

4.8 geomObjConicArc [104]

Overview

A child of geomObj. This object-type describes a curve that is defined by the equation

$$ax^2 + bxy + cy^2 + dx + ey + f = 0.$$

A conic arc can be either an ellipse, a hyperbola or a parabola. The object type is 104.

104 geomObjConicArc
double coefficients[11]
geomObjConicArc() void setCoefficient(int n, double value) double getCoefficient(int n)

Member functions

geomObjConicArc The constructor does some basic initializatons.

setCoefficient(int n, double value) This will set one of the 11 coefficients, which are described in table 4.1.

Coeff. #	Description
0	Conic Coefficient, a
1	Conic Coefficient, b
2	Conic Coefficient, c
3	Conic Coefficient, d
4	Conic Coefficient, e
5	Conic Coefficient, f
6	z coordinate of plane of definition
7	Start Point Abscissa
8	Start Point Ordinate
9	Terminate Point Abscissa
10	Terminate Point Ordinate

Table 4.1: Coefficients for the geomObjConicArc class.

double getCoefficient(int n) This method returns the requested coefficient. The coeffiecients are described in table 4.1.

Dependence on other classes

geomObjConicArc uses geomObj, color and OpenGL functions.

Example

See the source code for "Surfan".

4.9 geomObjPlane [108]

Overview

A child of geomObj. This object-type describes a curve that is defined by the equation

$$ax + by + cz = d.$$

The object type is 108.

108 geomObjPlane
geomObj *closingCurve double coefficients[4]
geomObjPlane() void setClosingCurve(geomObj *curve) void setCoefficient(int n, double value) geomObj *getClosingCurve() double getCoefficient(int n)

Member functions

geomObjPlane The constructor does some basic initializations.

void setClosingCurve(geomObj *curve) Specifies a geomObj that encloses the plane.

void setCoefficient(int n, double value) Allows the setting of the different plane coefficients a, b, c and d respectively.

geomObj *getClosingCurve() Returns a pointer to the geomObj that describes the circumference of the plane.

double getCoefficient(int n) Returns the plane coefficients.

Dependence on other classes

geomObjPlane is a child of geomObj. It also uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.10 geomObjLine [110]

Overview

A child of geomObj. This object-type describes an ordinary line in space, given by the parameterization

$$c(t) = p_1 + t(p_2 - p_1).$$

The object type is 110.

110 geomObjLine
coord start, end
void setStartPoint(coord start) void setEndPoint(coord end) coord getStartPoint() coord getEndPoint()

Member functions

geomObj The constructor does some basic initializations.

void setStartPoint(coord start) Specifies the starting point, p_1, of the line.

void setEndPoint(coord end) Specifies the end point, p_2, of the line.

coord getStartPoint() Returns p_1.

coord getEndPoint() Returns p_2.

Dependence on other classes

geomObjLine is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.11 geomObjParamSplineCurve [112]

Overview

A child of geomObj. This class describes a parametric spline curve divided into n segments. For $i = 1, \ldots, n$ the coordinates are given by

$$\begin{aligned} x(u) &= a_x x_i + b_x x_i s + c_x x_i s^2 + d_x x_i s^3 \\ y(u) &= a_y y_i + b_y x_i s + c_y y_i s^2 + d_y y_i s^3 \\ z(u) &= a_z z_i + b_z x_i s + c_z z_i s^2 + d_z z_i s^3 \end{aligned}$$

where (t_i are the breakpoints of the curve)

$$\begin{aligned} t_i \leq \quad u &\leq t_{i+1},\ i = 1, \ldots, n \\ s &= u - t_i \end{aligned}$$

The object type is 112.

112 geomObjParamSplineCurve
int noOfSegments int typeOrder int dimensions double *breakPoints double *parameters
geomObjParamSplineCurve(int degree, int segments) ~geomObjParamSplineCurve() void setBreakPoint(int n, double point) void setParameter(int n, double param) void setDimensions(int d) int getNoOfSegments() int getTypeOrder() int getDimensions() double getBreakPoint(int n) double getParameter(int n)

Member functions

geomObjParamSplineCurve(int segments) The constructor need to know how many segments the curve is made up of in order to allocate the correct amount of memory.

~geomObjParamSplineCurve() The destructor frees all allocated memory.

void setBreakPoint(int n, double point) Let's you specify the different break points.

void setParameter(int n, double param) Specifies the different parameters that characterize the curve. Their values and meanings are given in table 4.2.

Param. #	Description
0—$3, 12$—$12+3, 2\cdot 12$—$2\cdot 12+3, \ldots, (n-1)\cdot 12$—$(n-1)\cdot 12+3$	a_x, b_x, c_x and d_x
1—$4, 12+1$—$12+4, \ldots, (n-1)\cdot 12+1$—$(n-1)\cdot 12+4$	a_y, b_y, c_y and d_y
2—$5, 12+2$—$12+5, \ldots, (n-1)\cdot 12+2$—$(n-1)\cdot 12+5$	a_z, b_z, c_z and d_z
$1+13n$	x
$2+13n$	x'
$3+13n$	$x''/2!$
$4+13n$	$x'''/3!$
$5+13n$	y
$6+13n$	y'
$7+13n$	$y''/2!$
$8+13n$	$y'''/3!$
$9+13n$	z
$10+13n$	z'
$11+13n$	$z''/2!$
$11+13n$	$z'''/3!$

Table 4.2: Parameter specifications for geomObjParamSplineCurve.

void setDimensions(int d) Specifies if the curve is planar ,$d = 2$, or nonplanar, $d = 3$.

int getTypeOrder() Specifies the spline type of the curve. The different types are given but not explained in table 4.3.

Value	Spline Type
1	Linear
2	Quadratic
3	Cubic
4	Wilson-Fowler
5	Modified Wilson-Fowler
6	B-Spline

Table 4.3: The different spline types used for parametric spline curves or surfaces.

int getDimensions() Returns either 2 or 3 depending on whether the curve is planar or not.

double getBreakPoint(int n) Returns break point number n.

double getParameter(int n) Returns parameter value for parameter number n. The parameters are described in table 4.2.

Dependence on other classes

geomObjParamSplineCurve is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.12 geomObjParamSplineSurface [114]

Overview

A child of geomObj. This class describes a parametric spline surface given by the parameterisation

$$\begin{aligned}
x(u,v) &= a_{x_{i,j}} + b_{x_{i,j}} s + c_{x_{i,j}} s^2 + d_{x_{i,j}} s^3 + \\
&+ e_{x_{i,j}} t + f_{x_{i,j}} ts + g_{x_{i,j}} s^2 + h_{x_{i,j}} ts^3 + \\
&+ k_{x_{i,j}} t^2 + l_{x_{i,j}} t^2 s + m_{x_{i,j}} t^2 s^2 + n_{x_{i,j}} t^2 s^3 + \\
&+ p_{x_{i,j}} t^3 + q_{x_{i,j}} t^3 s + r_{x_{i,j}} t^3 s^2 + s_{x_{i,j}} t^3 s^3 \\
y(u,v) &= \ldots \\
z(u,v) &= \ldots
\end{aligned}$$

where

$$\begin{aligned}
T_{u_i} \leq \quad u \quad &\leq T_{u_{i+1}}, \quad i = 1, \ldots, M \\
s \quad = \quad & u - T_{u_i}
\end{aligned}$$

and

$$\begin{aligned}
T_{v_j} \leq \quad v \quad &\leq T_{v_{j+1}}, \quad j = 1, \ldots, N \\
s \quad = \quad & v - T_{v_j}
\end{aligned}$$

where T_u are the M breakpoints in the u-direction and T_v the N breakpoints in the v-direction. The object type is 114.

114 geomObjParamSplineSurface
int noUSegments, noVSegments int typeOrder double *breakPointsU double *breakPointsV double *parameters
geomObjParamSplineSurface(int degree, int Usegments, int Vsegments) ~geomObjParamSplineSurface() void setBreakPointU(int n, double point) void setBreakPointV(int n, double point) void setParameter(int n, double param) int getNoOfUSegments() int getNoOfVSegments() int getTypeOrder() double getBreakPointU(int n) double getBreakPointV(int n) double getParameter(int n)

Member functions

geomObjParamSplineSurface(int degree, int Usegments, int Vsegments) This is the constructor and it requires the degree of the surface as given by table 4.2 as well as the number of Usegments in the u-direction and the number of Vsegments in the v-direction.

~geomObjParamSplineSurface() The destructor frees all allocated memory.

void setBreakPointU(int n, double point) Sets the u-breakpoint n to the value point. n must lie between 0 and M, that is getNoOfUSegments().

void setBreakPointV(int n, double point) Sets the v-breakpoint n to the value point. n must lite between 0 and N, that is getNoOfVSegments()

void setParameter(int n, double param) Set parameter number n to param. The different parameter-meanings are given in table 4.4.

Param #	Description
$48 \cdot ((j-1)(N+1)+K-1)$	$a_{x_{j,k}}$, the first x-coefficient of the (j,k)-patch.
$48 \cdot ((j-1)(N+1)+K-1)+1$	$b_{x_{j,k}}$, the second x-coefficient of the (j,k)-patch.
$48 \cdot ((j-1)(N+1)+K-1)+16$	$a_{y_{j,k}}$, the first y-coefficient of the (j,k)-patch.
$48 \cdot ((j-1)(N+1)+K-1)+32$	$a_{z_{j,k}}$, the first z-coefficient of the (j,k)-patch.
$48 \cdot ((j-1)(N+1)+K)$	$s_{z_{j,k}}$, the first x-coefficient of the (j,k)-patch.

Table 4.4: The parameters used in geomObjParamSplineSurface.

int getNoOfUSegments() Returns the number of u-segments, M.

int getNoOfVSegments() Returns the number of v-segments, N.

int getTypeOrder() Returns the type of the spline as described in table 4.2.

double getBreakPointU(int n) Returns the nth break point in the u-direction.

double getBreakPointV(int n) Returns the nth break point in the v-direction.

double getParameter(int n) Returns the nth parameter.

Dependence on other classes

geomObjParamSplineSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.13 geomObjPoint [116]

Overview

A child of geomObj. This object describes a single point. The object type is 116.

106 geomObjPoint
geomObjPoint() void setPos(coord p) coord getPos()

Member functions

geomObjPoint() The constructor simply initiates the transformationmatrix et c.

void setPos(coord p) Set the point to be at position p.

coord getPos() Get the position of the point.

Dependence on other classes

geomObjPoint is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.14 geomObjRuledSurface [118]

Overview

A child of geomObj. This object type describes a surface formed by moving a line connecting points of equal relative arc length or equal relative parametric value on two parametric curves. If the two lines are given by $(c_{1_x}(t), c_{1_y}(t), c_{1_z}(t)),\ a \leq t \leq b$ and $(c_{2_x}(s), c_{2_y}(s), c_{2_z}(s)),\ c \leq s \leq d$, the coordinates of the points on the ruled surface can be written as

$$\begin{aligned} x(u,v) &= (1-v)c_{1_x}(t) + vc_{2_x}(s) \\ y(u,v) &= (1-v)c_{1_y}(t) + vc_{2_y}(s) \\ z(u,v) &= (1-v)c_{1_z}(t) + vc_{2_z}(s) \end{aligned}$$

where

$$\begin{aligned} 0 \leq & \quad u \quad \leq 1, \\ 0 \leq & \quad v \quad \leq 1, \\ t &= a + u(b-a) \\ s &= c + u(d-c), \text{if } \mathsf{direction} = 0 \\ s &= c + u(c-d), \text{if } \mathsf{direction} = 1 \end{aligned}$$

The object type is 118.

118 geomObjRuledSurface
geomObj *firstCurve, *lastCurve int direction
geomObjRuledSurface() void setFirstCurve(geomObj *curve) void setLastCurve(geomObj *curve) void setDirection(int dir) geomObj *getFirstCurve() geomObj *getLastCurve() int getDirection()

Member functions

geomObjRuledSurface() The constructor simply does some initializations.

void setFirstCurve(geomObj *curve) Specifies c_1.

void setLastCurve(geomObj *curve) Specifies c_2.

void setDirection(int dir) Sets which direction the surface should be drawn in, see the overview.

geomObj *getFirstCurve() Returns c_1.

geomObj *getLastCurve() Returns c_2.

int getDirection() Returns the direction.

Dependence on other classes

geomObjRuledSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for "Surfan".

4.15 geomObjRevolutionSurface [120]

Overview

A child of geomObj. This class describes an object that is constructed by rotating a curve around a specified axis. The class type is 120.

120 geomObjRevolutionSurface
vector axis geomObj *curve double startAngle, endAngle
geomObjRevolutionSurface() void setAxis(vector axis) void setCurve(geomObj *curve) void setStartAngle(double v) void setEndAngle(double v) vector getAxis() geomObj *getCurve() double getStartAngle() double getEndAngle()

Member functions

geomObjRevolutionSurface() The constructor does some necessary initializations.

void setAxis(vector axis) Sets the axis around which the curve is to be rotated.

void setCurve(geomObj *curve) Sets the curve that outlines the surface of revolution.

void setStartAngle(double v) Sets the start angle measured from the plane in which the curve is specified.

void setEndAngle(double v) Sets the end angle measured from the plane in which the curve is specified.

vector getAxis() Returns the axis of revolution.

geomObj *getCurve() Returns the curve.

double getStartAngle() Returns the start angle.

double getEndAngle() Returns the end angle.

Dependence on other classes

geomObjRevolutionSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.16 geomObjRatBSplineCurve [126]

Overview

A child of geomObj. This class describes a rational B-Spline Curve. The object type is 126

126 geomObjRatBSplineCurve
int form int upperIndex int degree bool planar, closed, rational, periodic double startParVal, endParVal vector normal double *knots double *weights coord *controlPoints
geomObjRatBSplineCurve(int upindex, int degree) ~geomObjRatBSplineCurve() void setKnot(int n, double data) void setControlPoint(int n, coord p) void setWeight(int n, double w) void setNormal(vector n) void setPlanar(bool tf) void setClosed(bool tf) void setRational(bool tf) void setPeriodic(bool tf) void setForm(int f) int getDegree() bool isPlanar() bool isClosed() bool isRational() bool isPeriodic() coord getControlPoint(int n) double getWeight(int n) double getKnot(int n) vector getNormal() int getForm()

Member functions

geomObjRatBSplineCurve(int upindex, int degree) The constructor needs to know how many breakpoints and weights there are (upindex) as well as the degree of the basis functions.

~geomObjRatBSplineCurve() The destructor frees all allocated memory.

void setKnot(int n, double data) Sets the n:th element in the knot vector to data.

void setControlPoint(int n, coord p) Specifies the n:th control point, which is set to p.

void setWeight(int n, double w) Set the weight associated with the n:th control point.

void setNormal(vector n) If the curve is planar, a vector normal to the plane it lies in can be specified.

void setPlanar(bool tf) Specifies if it is a planar curve or not.

void setClosed(bool tf) Specifies if the curve is closed or not.

void setRational(bool tf) Specifies if the curve is rational or polynomial.

void setPeriodic(bool tf) Specifies if the curve is periodic or not.

void setForm(int f) Set the form of the curve (unnecessary?) according to table 4.5

Form	Description
0	Form of curve must be determined from the rational B-spline parameters.
1	Line
2	Circular arc
3	Elliptical arc
4	Parabolic arc
5	Hyperbolic arc

Table 4.5: The different form types for geomObjRatBSplineCurve.

int getDegree() Returns the degree of the basis functions.

bool isPlanar() Returns TRUE if curve is planar, otherwise FALSE.

bool isClosed() Returns TRUE if curve is closed, otherwise FALSE.

bool isRational() Returns TRUE if curve is rational, otherwise FALSE.

bool isPeriodic() Returns TRUE if curve is periodic, otherwise FALSE.

coord getControlPoint(int n) Returns the n:th control point.

double getWeight(int n) Returns the n:th weight.

double getKnot(int n) Returns the n:th element of the knotvector.

vector getNormal() If the curve is planar, this function returns the normal of the curve, otherwise it returns the null-vector, $(0, 0, 0)$.

int getForm() Returns the form type of the curve. Confer table 4.5.

Dependence on other classes

geomObjRatBSplineCurve is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.17 geomObjRatBSplineSurface [128]

Overview

A child of geomObj. This class describes a rational B-spline surface. The object type is 128.

128 geomObjRatBSplineSurface
int form int upIndex1, upIndex2 int degree1, degree2 bool closed1, closed2, rational, periodic1, periodic2 double *knots1, *knots2, *weights coord *controlPoints double start1, end1, start2, end2
geomObjRatBSplineSurface(int upindex1, int upindex2, int degree1, int degree2) ~geomObjRatBSplineSurface() void setForm(int f) void setClosed1(bool tf) void setClosed2(bool tf) void setRational(bool tf) void setPeriodic1(bool tf) void setPeriodic2(bool tf) void setControlPoint(int n, coord p) void setKnot1(int n, double k) void setKnot2(int n, double k) void setParDir1(double start, double end) void setParDir2(double start, double end) void setWeight(int n, double w) int getDegree1() int getDegree2() bool isRational() bool isClosed1() bool isClosed2() bool isPeriodic1() bool isPeriodic2() double getKnot1(int n) double getKnot2(int n) double getWeight(int n) int getUpIndex1() int getUpIndex2() coord getControlPoint(int n)

Member functions

geomObjRatBSplineSurface(int upindex1, int upindex2, int degree1, int degree2)
The constructor needs to know the upper index of the two sums, upindex1 and upindex2, as well as the degrees of the two sets of basis functions, degree1 and degree2.

~geomObjRatBSplineSurface() The destructor frees all allocated memory.

void setForm(int f) Sets the form type of the surface according to the types given in table 4.6

Form	Description
0	Form of the surface must be determined from the rational B-spline parameters
1	Plane
2	Right circular cylinder
3	Cone
4	Sphere
5	Torus
6	Surface of revolution
7	Tabulated cylinder
8	Ruled surface
9	General quadratic surface

Table 4.6: The different form types for geomObjParamBSplineSurface.

void setClosed1(bool tf) Specifies whether the surface is closed (TRUE) in the first parametric direction or not (FALSE).

void setClosed2(bool tf) Specifies whether the surface is closed (TRUE) in the second parametric direction or not (FALSE).

void setRational(bool tf) Specifies whether the surface is rational (TRUE) or polynomial (FALSE).

void setPeriodic1(bool tf) Specifies whether the surface is periodic (TRUE) in the first parametric direction or not (FALSE).

void setPeriodic2(bool tf) Specifies whether the surface is periodic (TRUE) in the second parametric direction or not (FALSE).

void setControlPoint(int n, coord p) Sets the position of the n:th control-point to p.

void setKnot1(int n, double k) Sets the n:th element of the first knotvector.

void setKnot2(int n, double k) Sets the n:th element of the second knotvector.

void setParDir1(double start, double end) Sets the starting and the ending values for the first parametric direction.

void setParDir2(double start, double end) Sets the starting and the ending values for the second parametric direction.

void setWeight(int n, double w) Set the weight associated with the n:th control point.

int getDegree1() Returns the degree of the first set of basis functions.

int getDegree2() Returns the degree of the second set of basis functions.

bool isRational() Returns TRUE if the surface is rational, otherwise FALSE.

bool isClosed1() Returns TRUE if the surface is closed in the first parametric direction, otherwise FALSE.

bool isClosed2() Returns TRUE if the surface is closed in the second parametric direction, otherwise FALSE.

bool isPeriodic1() Returns TRUE if the surface is periodic in the first parametric direction, otherwise FALSE.

bool isPeriodic2() Returns TRUE if the surface is periodic in the second parametric direction, otherwise FALSE.

double getKnot1(int n) Returns the n:th element of the first knotvector.

double getKnot2(int n) Returns the n:th element of the second knotvector.

double getWeight(int n) Returns the n:th weight.

int getUpIndex1() Returns the upper index of the first sum.

int getUpIndex2() Returns the upper index of the second sum.

coord getControlPoint(int n) Returns the coordinates of the n:th controlpoint.

Dependence on other classes

geomObjRatBSplineSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for "Surfan".

4.18 geomObjCurveParamSurface [142]

Overview

A child of geomObj. This class describes a curve on a parametric surface. If $S = S(u,v) = (z(u,v), y(u,v), z(u,v))$ is a regular parametric surface on the rectangular domain $D = \{(u,v) : u_1 \le u \le u_2,\ v_1 \le v \le v_2\}$ and $B = B(t) = (u(t), v(t)),\ a \le t \le b$ is a curve taking its values in D. Then the curve on S is defined as follows

$$\begin{aligned} C(t) &= S \circ B(t) = S(B(t)) = S(u(t), v(t)) = \\ &= (x(u(t), v(t)), y(u(t), v(t)), z(u(t), v(t)))\ ,\ a \le t \le b. \end{aligned}$$

The object type is 142.

142 geomObjCurveParamSurface
geomObj *surface
geomObj *curveOnSurface
geomObj *curve
geomObjCurveParamSurface()
void setSurface(geomObj *surface)
void setCurveOnSurface(geomObj *curve)
void setCurve(geomObj *curve)
geomObj *getSurface()
geomObj *getCurveOnSurface()
geomObj *getCurve()

Member functions

geomObjCurveParamSurface() The constructor just does some basic initializations.

void setSurface(geomObj *surface) Specifies the surface S.

void setCurveOnSurface(geomObj *curve) Specifies the curve B.

void setCurve(geomObj *curve) Specifies the curve C.

geomObj *getSurface() Returns a pointer to the surface S.

geomObj *getCurveOnSurface() Returns a pointer to the curve B.

geomObj *getCurve() Returns a pointer to the curve C.

Dependence on other classes

geomObjCurveParamSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for "Surfan".

4.19 geomObjTrimmedSurface [144]

Overview

A child of geomObj. This class describes a surface, D, that has been trimmed by an outer boundary that lies in D and can, but doesn't have to, be the boundary of D and several inner boundaries which are curves that are mutually disjoint and lies within the interior of the outer boundary. The object type is 144.

144 geomObjTrimmedSurface
geomObj *surface int numberOfCurves geomObjCurveParamSurface *outerCurve geomObjCurveParamSurface **innerCurves
geomObjTrimmedSurface(int noCurves) ~geomObjTrimmedSurface() void setSurface(geomObj *sur) void setInnerCurve(int n, geomObjCurveParamSurface *curve) void setOuterCurve(geomObjCurveParamSurface *curve) geomObjCurveParamSurface *getInnerCurve(int n) geomObjCurveParamSurface *getOuterCurve() geomObj *getSurface() int getNoOfCurves()

Member functions

geomObjTrimmedSurface(int noCurves) The constructor needs to know the number of inner boundary curves, noCurves.

~geomObjTrimmedSurface() The destructor frees all allocated memory.

void setSurface(geomObj *sur) Specifies D, the surface.

void setInnerCurve(int n, geomObjCurveParamSurface *curve) Specifies the n:th inner boundary curve to be curve.

void setOuterCurve(geomObjCurveParamSurface *curve) Specifies the outer boundary curve to be curve.

geomObjCurveParamSurface *getInnerCurve(int n) Returns a pointer to the n:th inner boundary curve.

geomObjCurveParamSurface *getOuterCurve() Returns a pointer to the outer boundary curve.

geomObj *getSurface() Returns a pointer to the surface.

int getNoOfCurves() Returns the number of inner boundary curves.

Dependence on other classes

geomObjTrimmedSurface is a child of geomObj and uses OpenGL functions and the color class.

Example

See the source code for “Surfan”.

4.20 graphHandler

Overview

The graphHandler class takes care of everything that has to do with the connectivity graph; loading and saving it, drawing it, picking nodes, adding or removing edges.

graphHandler
graphHandler(Widget) void makeGraph(objHandler *) void flipDrawGraph() void drawTheGraph() void drawNodesWithPickInfo() void loadGraph(char *) void saveGraph(char *) void trigNode(int) void increaseNodes() void decreaseNodes()

Member functions

graphHandler(Widget) The constructor simply initializes a few class members. The provided widget is one that can act as a parent to graphHandler's popupWindows; since the graphHandler doesn't know whether the window calling it is open, the constructor is called with the main window widget today.

void makeGraph(objHandler *) Given information about an object (the objHandler pointer), this function creates a graph consisting of all the nodes corresponding to the patches of the object.

void flipDrawGraph() Switch between drawing and not drawing the graph.

void drawTheGraph() Draw the graph.

void drawNodesWithPickInfo() Draw only the nodes of the graph. This is used when picking objects using OpenGL:s selection mechanism.

void loadGraph(char *)

void saveGraph(char *)

void trigNode(int) The node with the given number has been selected. If this node was already selected, this function deselects it. If another node was already selected, that node is deselected and an edge is added or remopved between these two edges. If no node was selected before, this node is highlighted.

void increaseNodes() Increase the size of the nodes.

void decreaseNodes() Decrease the size of the nodes.

Dependance on other classes

The graphHandler class uses an object of type popupWindows. It also takes an object of type objHandler to make the connectivity graph.

Example

See the source code for "Surfan".

4.21 mainWin

Overview

text

mainWin
mainWin(Widget) void createMenu() void setTriangulationWindow(triWin *) void setSurfaceWindow(surfWin *) Widget getWidget()

Member functions

mainWin(Widget) Sets up the main window.

void createMenu() Sets up the menu of the main window.

void setTriangulationWindow(triWin *) Give the main window a pointer to the current triangulation window

void setSurfaceWindow(surfWin *) Give the main window a pointer to the current surface window

Widget getWidget() Return the main window widget.

Dependance on other classes

mainWin uses an object of type menu.

Example

See source code for “Surfan”

4.22 menu

Overview

The class menu provides a simple way to incorporate a menu in an application. In the constructor, an integer is given, representing the number of menu panes that the menu should have. If more menu panes than this are being inserted, the program will simply skip the last ones. When a menu pane is determined, using addMenuPane or addHelpMenuPane, its entries are added using addMenuItem.

menu
menu(Widget,int) Widget getWidget() void addMenuPane(char *) void addHelpMenuPane(char *) void addMenuItem(char *,XtCallbackProc,XtPointer) void addSeparator()

Member functions

menu(Widget,int) The constructor takes a parent widget as its first argument. This is the widget that "owns" the menu. The second argument is an integer representing the number of menu panes in the menu.

Widget getWidget() returns the menu bar widget.

void addMenuPane(char *) This member function adds a menu pane with the supplied name.

void addHelpMenuPane(char *) This member works just like the above one, but aligns the menu pane to the right in the menu bar.

void addMenuItem(char *,XtCallbackProc,XtPointer) This function adds an entry into the current menu pane (the one that was last added). The name of this entry is the first parameter, a pointer to a callback procedure is the second and a pointer to the data that should be supplied to the callback is the third.

void addSeparator() This function adds a separator in the current pane.

Dependance on other classes

None.

Example

```
#include <iostream.h>    // For the stream output
#include "menu.h"        // Include the menu class

void quitCB(Widget w,XtPointer clientData,XtPointer callData){
  exit(0);
}

void choiceOneCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 1" << endl;
}

void choiceTwoCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 2" << endl;
}

void choiceThreeCB(Widget w,XtPointer clientData,XtPointer callData){
  cout << "Choice: 3" << endl;
}

void setUpTheMenus(Widget parent){
  menu theMenu(parent,2);
  theMenu.addMenuPane("OneTwo");
  theMenu.addMenuItem("Choice 1",choiceOneCB,NULL);
  theMenu.addMenuItem("Choice 2",choiceTwoCB,NULL);
  theMenu.addSeparator();
  theMenu.addMenuItem("Quit",quitCB,NULL);
  theMenu.addHelpMenuPane("Three");
  theMenu.addMenuItem("Choice 3",choiceThreeCB,NULL);
}

void main(int argc,char **argv){
  Widget toplevel;
  XtAppContext app;

  toplevel = XtAppInitialize(&app,"Test",NULL,0,&argc,argv,NULL,NULL,0);

  XtVaSetValues(toplevel, XmNwidth, 400, NULL);

  setUpTheMenus(toplevel);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app);
}
```

4.23 model

Overview

This class handles an "object", i.e. a CAD model, and additional information, e.g. graph information, triangulation etc. A coord is a struct (with double x, y, z) representing a coordinate. drawMode can be either of polygons, coloredPolygons, transparentPolygons and wireFrame.

model
model() void loadIges(char *) void loadGraph(char *) void saveGraph(char *) void draw() void drawTriangulation() void drawTheGraph() void drawNodesWithPickInfo() void setDrawMode(drawMode) void flipDrawGraph() void trigNode(GLuint) void increaseNodes() void decreaseNodes() Boolean getListsExist() void setListsExist(Boolean) double getSize() coord *getCenter()

Member functions

model() The constructor creates an object and sets up some basic information about the model.

void loadIges(char *) Given a file name, this function loads an IGES file into memory.

void loadGraph(char *) Given a file name, this function loads a graph file into memory, by calling the loadGraph function in graphHandler.

void saveGraph(char *) Given a file name, this function saves a graph file to disk, by calling the saveGraph function in graphHandler.

void draw() Draw this object.

void drawTriangulation() Draws the triangulation of the model.

void drawTheGraph() Draws the connectivity graph of the object.

void drawNodesWithPickInfo() Draw the nodes with picking information. This is used with OpenGL's picking functions, see [21].

void setDrawMode(drawMode) Set drawing mode. The drawing mode is of type drawMode and can have the values polygons (all patches are drawn as polygons, using one colour), coloredPolygons (all patches are drawn using different colours to make them easy to determine from each other), transparentPolygons (the polygons are drawn semi-transparent so that the nodes are visible), or wireFrame (only the outlines of the patches are drawn).

void drawTheGraph() Draws the graph that corresponds to this model.

void drawNodesWithPickInfo() Draw only the nodes of the graph. This is used when picking objects using OpenGL:s selection mechanism.

void flipDrawGraph() Turn the drawing of a graph on or off.

void trigNode(GLuint) The node with the given number has been selected. If this node was already selected, this function deselects it. If another node was already selected, that node is deselected and an edge is added or remopved between these two edges. If no node was selected before, this node is highlighted.

void flipDrawGraph() This functions toggles between drawing and not drawing the graph.

void trigNode(GLuint) This function "trigs" a node, and is called when the user clicks on a node while keeping the "Ctrl" button depressed.

void increaseNodes() Increase the size of the nodes.

void decreaseNodes() Decrease the size of the nodes.

Boolean getListsExist() Tells whether the display lists (see [21]) of the model exists.

void setListsExist(Boolean) Tell the program that the lists do exist.

double getSize() This function returns the size of the object.

coord *getCenter() This function returns the coordinates of the center of the object.

Dependance on other classes

This class uses objects of classes ModelIO, objHandler, graphHandler and triHandler.

Example

See the source code for "Surfan".

4.24 ModelIO

Overview

ModelIO contains methods to read and write objHandler data using primarily the IGES format. VDA will later be supported.

ModelIO
objHandler *readIGES(char *filename)

Member functions

objHandler *readIGES(char *filename) This method will read the IGES-file specified by its filename.

Dependence on other classes

This class uses LEDA list and GRAPH classes, and it returns an objHandler *.

Example

The following example is very simple and selfexplanatory. It simply reads IGES-data and does nothing with it.

```
#include "ModelIO.h"
#include "objHandler.h"

void main(int argc, char *argv[]) {

  ModelIO fileIO;
  objHandler *theModel;

  theModel = fileIO.readIGES("test.igs");

}
```

4.25 objHandler

Overview

objHandler manages the tree that describes all the objects in a particular model.

objHandler
GRAPH¡geomObj *, int¿ objTree
list¡geomObj *¿ objOrderList
objHandler()
˜objHadler()
geomObj *rootObj()
geomObj *nextObj()
geomObj *prevObj()
geomObj *thisObj()
geomObj *getIdObj(int id)
int getObjId(geomObj *)
void addObj(geomObj *obj, int id)
void addEdge(int idfrom, int idto)
void addEdgeObj(geomObj *from, geomObj *to)
geomObj *existsObj(int id, geomObj *parent)
int noInEdges(int id)
int noOfObjs()
void draw(bool changeColor, color *col)
double getSize()
vectorPtr getCenter()

Member functions

objHandler() The constructor does some basic initializing and creates the root object which is an entity transformation matrix.

˜objHadler() The destructor frees all the memory, including all the geomObjs.

geomObj *rootObj() Move to root of object-tree. Retrieve this particular object.

geomObj *nextObj() Move to the next object using the order in objOrderList. A depth first pattern is used. Retrieve this particular object.

geomObj *prevObj() Move to, and retrieve, the previous object in the objOrderList.

geomObj *thisObj() Which object am I?

geomObj *getIdObj(int id) Returns the geomObj associated with a certain id.

int getObjId(geomObj *) Returns the id associated with a certain geomObj.

void addObj(geomObj *obj, int id) Adds the geomObj obj with the unique id to objTree.

void addEdge(int idfrom, int idto) Adds a directed edge in objTree between idfrom and idto.

void addEdgeObj(geomObj *from, geomObj *to) Adds a directed edge in objTree between from and to.

geomObj *existsObj(int id, geomObj *parent) Checks if the geomObj with id id and parent parent. If it exists, id's associated geomObj will be returned, otherwise NULL.

int noInEdges(int id) Returns the number of edges that end in the geomObj associated with id.

int noOfObjs() Returns the total number of geomObjs in objTree.

void draw(bool changeColor, color *col) Calls draw of the root-object.

double getSize() Returns the approximate total size of all objects in objTree.

vectorPtr getCenter() Returns a vector pointing to the center of all objects in objTree.

Dependence on other classes

objHandler uses LEDA's list and GRAPH classes as well as geomObj.

Example

4.26 popupWindows

Overview

This class displays all the information, help and error messages that may appear. It is initialized with a widget that is guaranteed always to be realized when one of this class's functions is called.

popupWindows
popupWindows(Widget) void errorWindow(String) void helpWindow(String) void infoWindow(String)

Member functions

popupWindows(Widget) Initialize the class with a parent widget that is guaranteed always to be realized when one of this class's functions is called.

void errorWindow(String) Popup an error window showing the given string.

void helpWindow(String) Popup a help window showing the given string.

void infoWindow(String) Popup an information window showing the given string.

Dependance on other classes

None

Example

See the source code for "Surfan".

4.27 surfWin

Overview

This class implements the "Model and Graph" window.

surfWin
surfWin(Widget,String,int,int,int,int) void setup(Widget) void setModel(model *) void loadIges(char *) void loadGraph(char *) void saveGraph(char *) void draw() void draw(Widget) void resize(Widget) void resetView() void keyPressed(Widget,XEvent *) void keyReleased(Widget,XEvent *) void getMouse(Widget,XEvent *) void trackMouse(Widget,XEvent *) void releaseMouse(Widget,XEvent *) int determineButton(int)

Member functions

surfWin(Widget,String,int,int,int,int) The constructor initializes the surface window. The first two parameters are the widget that should act as a parent to the surface window, and the name of the surface window. The four integers determine the x position, y position, width and height of the window respectively.

void setup(Widget) This function sets up the surface window. Call it after creating the surface window. The widget is a widget that can act as parent to all surfWin's popupWindows.

void setModel(model *) Give the surface window a pointer to the current model.

void loadIges(char *) Load a model file using the IGES format, by calling model's loadIges() function.

void loadGraph(char *) Load a graph using the GML format, by calling model's loadGraph() function.

void saveGraph(char *) Save a model using the GML format, by calling model's saveGraph() function.

void resetView() Reset the viewing position to its original value.

draw() Draw the object. This function merely calls draw(Widget) with a given widget.

void draw(Widget) Draw the object. This function overrides that of dispWin.

void resize(Widget) This function is automatically called when the window is resized.

void keyPressed(Widget,XEvent *) This function is called whenever a key is pressed. The XEvent struct contains information about which key it was.

void keyReleased(Widget,XEvent *) This function is called whenever a key is released. The XEvent struct contains information about which key it was.

void getMouse(Widget,XEvent *) This function is called whenever a mouse button is pressed. Use the XEvent struct to get additional information.

void trackMouse(Widget,XEvent *) This function is called whenever the mouse is moved. Use the XEvent struct to get additional information.

void releaseMouse(Widget,XEvent *) This function is called whenever a mouse button is released. Use the XEvent struct to get additional information.

int determineButton(int) Given an integer, corresponding to the xbutton.button entry in an XEvent struct, this function returns 0 if the left button is pressed, 1 if it is the moddle button and 2 if it is the right button.

Dependance on other classes

surfWin is derived from dispWin and uses object of types model and popupWindows.

Example

See source code for "Surfan".

4.28 triHandler *preliminary!*

Overview

This class will be for a triangulation what graphHandler is for a connectivity graph, cf graphHandler.

triHandler
draw()

Member functions

draw() Draw the triangulation.

Dependance on other classes

Example

4.29 triWin

Overview

The triWin class handles the triangulation window. It automatically handles all the menus and the mouse and keyboard control.

triWin
triWin(Widget shell,String name,int x,int y,int w,int h) void setup() void setModel(model *) void draw() void draw(Widget) void resize(Widget) void resetView() void keyPressed(Widget,XEvent *) void keyReleased(Widget,XEvent *) void getMouse(Widget,XEvent *) void trackMouse(Widget,XEvent *) void releaseMouse(Widget,XEvent *) int determineButton(int)

Member functions

triWin(Widget,String,int,int,int,int) The constructor initializes the triangulation window. The first two parameters are the widget that should act as a parent to the triangulation window, and the name of the triangulation window. The four integers determine the x position, y position, width and height of the window respectively.

void setup() This function sets up the triangulation window. Call it after creating the triangulation window.

void setModel(model *) Give the triangulation window a pointer to the current model.

void draw() Draw the object.

void draw(Widget) Call draw(). This function is here only to make this derived class comply with its base class dispWin.

void resize(Widget) This function is automatically called when the window is resized.

void resetView() Reset the viewing position to its original value.

void keyPressed(Widget,XEvent *) This function is called whenever a key is pressed. The XEvent struct contains information about which key it was.

void keyReleased(Widget,XEvent *) This function is called whenever a key is released. The XEvent struct contains information about which key it was.

void getMouse(Widget,XEvent *) This function is called whenever a mouse button is pressed. Use the XEvent struct to get additional information.

void trackMouse(Widget,XEvent *) This function is called whenever the mouse is moved. Use the XEvent struct to get additional information.

void releaseMouse(Widget,XEvent *) This function is called whenever a mouse button is released. Use the XEvent struct to get additional information.

int determineButton(int) Given an integer, corresponding to the xbutton.button entry in an XEvent struct, this function returns 0 if the left button is pressed, 1 if it is the middle button and 2 if it is the right button.

Dependance on other classes

triWin is derived from dispWin and uses objects of types model and popupWindows.

Example

See source code for "Surfan".

Chapter 5

GUI design issues

5.1 The nodeInfo structure

For building a connectivity graph, we use the LEDA [18] type for parameterized graphs. Its node parameter is a pointer to a nodeInfo structure, and its edge info is an integer. The edge info is never used. The nodeInfo class has five entries: Boolean highlighted that tells whether a node is highligted (selected during picking, see 2.3) or not, int identity that is a unique identity number to use with the LEDA operations, and double x,y,z that are the coordinates of the node.

5.2 Adding a menu item/callback function

To add a menu item to an existing menu, there are a few steps to consider. Assume that we want to add a menu item to triWin, the triangulation window. Assume also that we have an integer variable, infoInt, in the triWin class, that contains some information about the instance of the class. When selecting "Show Info" that exists under the "File" menu, this integer is printed on standard out. The triWin class is inherited from dispWin and therefore inherits all dispWin's methods for handling a menu. The "File" menu already exists in the triWin class, so all we have to do is add the item, and a function that will be called when this item is selected. This function is a so called callback function, and if it is part of the triWin class it must be declared static (see [26]) in triWin.h:

```
static void printInfoCallback(Widget w,XtPointer clientData,
                              XtPointer callData);
```

The implementation would look something like:

```
void printInfoCallback(Widget w,XtPointer clientData,
```

```
                          XtPointer callData){
    triWin *obj = (triWin *) clientData;
    cout << obj->infoInt << endl;
  }
```

Since the function is declared static, there exists only one instance of it, no matter how many instances of the triWin class we have. To know which instance is calling, the function has the line triWin *obj = (triWin *) clientData to determine this. This means that when we set up the caller (i.e. the menu alternative), we must provide information about which instance is calling printInfoCallback. This we do by sending a this pointer as client data. The following lines already exist in triWin.cpp:

```
addMenuPane(``File'');
addMenuItem(``Close'', &triWin::closeCB, (XtPointer) this);
```

To add another menu alternative, we simply insert an appropriate line:

```
addMenuPane(``File'');
addMenuItem(``Show Info'', &triWin::printInfoCallback,
            (XtPointer) this);
addMenuItem(``Close'', &triWin::closeCB, (XtPointer) this);
```

This declaration of a callback function is applicable also in other cases, where the callback isn't the result of selecting a menu option.

Chapter 6

ModelIO, objHandler and geomObj design issues

The purpose of this chapter is to describe how the classes ModelIO, objHandler and geomObj works and what was the intentions and meanings of what was implemented.

6.1 ModelIO operations

The only presently available method in ModelIO is readIGES . It takes as its argument a filename and returns a pointer to an objHandler instance.

6.1.1 readIGES

This method reads the IGES-file step by step using the following functions:

isIGESFile Tries using a simple check of the 73rd character of every line in the file whether or not it is a valid IGES-file. This function uses readIGESNextLine for reading the file.

readIGESComment A comment has a 'S' for Start section in the 73rd position. This function reads all those lines from the IGES-files and returns them in a character array. For reading it uses readIGESNextLine .

readIGESGlobal This function parses the 'G' section of the IGES-file. It uses the function readIGESNextLine to read the whole section into a single variable (full) which is then fed to readIGESStr/Int/RealPar respectively. Since the look of the global section is known and inflexible, we know before-hand what is coming.

The data in the global section is stored in the array IGESGlobalData which is presently not used elsewhere.

readIGESDirectory After the global section, the directory, 'D', section is read. The function uses readIGESNextLine (how original) for retrieving the data which it uses two build the IGESDataList which later also will contain all the parametric data for each entity. It separates the ID of each entity as its 'D'-line number, it seperates the TYPE and lastly it determines whether the entity is visible or not. Or rather if it should be directly visible in objHandler.

Whether or not an entry is to be included is determined by a call to validIGESEntity which uses the array IGESEntityForm to decide this. IGESEntityForm also includes information about how the parametric data is stored as described in table 6.1.

Entry	Type of data	Description
type	int	An IGES type number such as for example 126.
data	char *	Describes how the data is stored in the IGES-file. 'r' stands for a double, 'i' for an int 'p' for a pointer (int) and 's' for a string. If the entity in the file contains more parameters than what is listed, the last parameter will be used for the last parameters.
func	Function pointer	Points to a function that converts the parameters to a geomObj.

Table 6.1: Description of the dataformat of ModelIO's IGESEntityForm

After the entire parameter section has been read, IGESDataList is sorted according to the order of the entities in the parameter section. This enables the whole IGES-file to be read in only one pass (actually two passes are used, the first when determining whether or not the file is an IGES-file).

readIGESParameter This simple function simply goes through IGESDataList and calls readIGESParameters for every item in it.

readIGESParameters First, all the parameters for a particular entity is read using readIGESRecord (which also parses the parameters some). Then the parameters are parsed using IGESEntityForm and readIGESStr/Int/RealPar respectively. addIGESParameter is used to allocate the memory.

buildIGESObjHandlerTree Lastly, the objHandler tree is built. First objHandler is initiated and the root-object is retrieved (which is automatically generated to an entity transformation matrix in the constructor of objHandler). Then, all visible entities in IGESDataList that have not yet been added and are not transformation matrices (IGES type 124) are added and set to be children of the root.

The entities are added using addIGESObject which uses IGESEntityForm to call the correct functions for adding the entities, checks if the entity has a transformationmatrix in which case it adds it (using addIGESTrans). addIGESObject also describes the relationships between the different entities.

6.1.2 Adding new IGES-entity types

Care has been taken so that new IGES entities can easily be added. In ModelIO, new code has to be inserted in two places, IGESEntityForm where its format is described, and then the function that moves the data to geomObj.

A typical IGESEntityForm entry (this is for the Point entity) is shown in figure 6.1.

```
116, "rrrp", &ModelIO::IGESPoint,
```

Figure 6.1: A sample entry in ModelIO's IGESEntityForm

The corresponding ModelIO::IGESPoint is shown in figure 6.2. Mandatory function calls in this function are `ob = new geomObjPoint();` (depending, of course, on the entity type), `obj->addObj(ob, ptr->id);` which ought to be the same for all object types and `ob->done();` if post-processing of the data is to be done by geomObj .

```
geomObj *ModelIO::IGESPoint(IGESDataPtr ptr, objHandler *obj,
                            ModelIO *) {

  geomObjPoint *ob;
  coord tmp;

  ob = new geomObjPoint();
  obj->addObj(ob, ptr->id);

  tmp.x = ptr->parameters[0].d;
  tmp.y = ptr->parameters[1].d;
  tmp.z = ptr->parameters[2].d;
  ob->setPos(tmp);

  ob->done();
  return ob;
}
```

Figure 6.2: An example of a ModelIO-function to move data to a geomObj.

Depending on the desired level of data-abstraction a new geomObj sub-class has to be constructed as well. For example, a circular arc can be converted into a NURB at this stage removing the necessity of geomObjCircularArc. On the other hand, that removes knowledge of the heritage of the geomObj.

6.2 objHandler

objHandler works as a wrapper for geomObj . The instances of geomObj know about each other through the transformation matrixes and the parent pointer. Through a LEDA GRAPH in the form of a tree, objHandler knows of these relationships and is there by allowed to perform different search operations on the tree. Presently it is possible to go through all the "real" objects in the tree using a depth first search pattern. A simple tree is shown in figure 6.3.

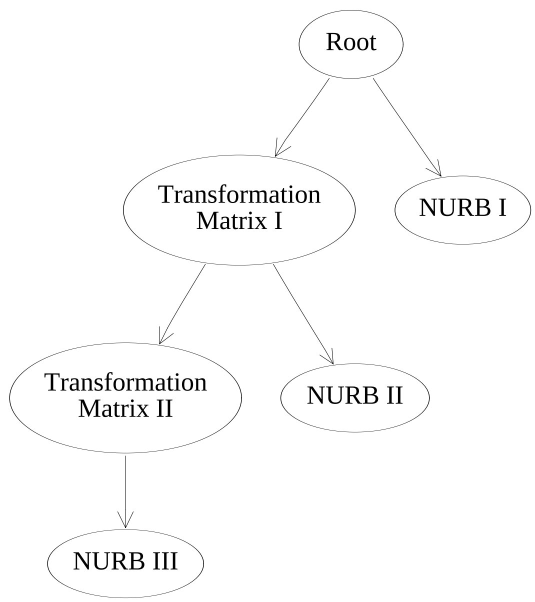

Figure 6.3: A tree of objects in objHandler. (GraphViz)

In objHandler there are three prominent data structures:

objTree is a LEDA graph only containing geomObj *.

objOrderList is a list containing the traversal order of the nodes in objTree. A call to getRootObj initializes the list in a depth first search using LEDA routines. Subsequent calls to getNextObj , getPrevObj and getThisObj then traverses this list. All transformation matrices (geomObj type 0) are filtered by these methods.

objIds is a list that contains all the geomObjs with some additional information about them such as an Id and their node in objTree. This is primarily used when constructing objTree.

Apart from this objHandler also contains a lot of functions used when constructing the tree (add* functions). draw will draw the entire tree (it calls the draw function of the root).

6.3 geomObj

The different geomObjs are related through the parent variable. The only objects that has a "downward" knowledge of the others are the composite object-types and the transformation matrixes.

The done method is ment to be used to prepare data for drawing, so that it can be drawn a little bit faster. As of now, all drawing is made using OpenGL routines. In table 6.2 the status of the current more-or-less implemented object-types.

Type	Name	Status
0	geomObjTransMatrix	Done, seems to be working.
100	geomObjCircularArc	Done, though there seems to be trouble with the drawing.
102	geomObjCompositeCurve	Done, seems to be working.
104	geomObjConicArc	All but drawing done.
108	geomObjPlane	All but drawing done.
110	geomObjLine	Done, seems to be working.
112	geomObjParamSplineCurve	Done, though untested.
114	geomObjParamSplineSurface	Done, though untested.
116	geomObjPoint	Done, though untested.
118	geomObjRuledSurface	All but drawing done.
120	geomObjRevolutionSurface	Done for when the curve is a NURB (untested), otherwise all except drawing.
126	geomObjRatBSplineCurve	Done, though untested.
128	geomObjRatBSlineSurface	Done, seems to be working.
142	geomObjCurveParamSurface	All but drawing done.
144	geomObjTrimmedSurface	All but drawing done.

Table 6.2: Current status of geomObj.

The implementation of the different objects should be pretty straight-forward to understand, though note that the trMatrix data type is the same as is used for transformation matrices in OpenGL (double[16]),

$$\mathsf{trMatrix} = \begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix},$$

where i is the index in the array.

Chapter 7

Source Code Structure

7.1 Directories

```
SurfaceAnalyzer
 |- Bin
 |- Doc
 |    |- Appendix
 |    |- Implementation
 |    |    |- Classes
 |    |    |- Figures
 |    |         |- Xfig
 |    |- Theory
 |    |- Usersguide
 |         |- Figures
 |              |- Xfig
 |- Lib
 |- Obj
 |- Src
      |- Include
           |- LEDA
```

7.2 Files

Bin

README Information about this directory.

Doc

doc.tex The main documentation file. This file includes all other doc files.

future.tex Describes future enhancements for Surfan.

makefile Creates the documentation.

references.tex The reference list for Surfan

Doc/Appendix

iges.tex Contains definitions of all implemented IGES entities.

Doc/Implementation

files.tex Describes the file structure.

overview.tex Overall implementation notes.

Doc/Implementation/Classes

basewin.tex Describes the baseWin class.

classdoc.tex Overview of the class descriptions.

color.tex Describes the color class.

dispwin.tex Describes the dispWin class.

geomobj.tex Describes the geomObj* classes.

graphHandler.tex Describes the graphHandler class.

mainwin.tex Describes the mainWin class.

menu.tex Describes the menu class.

model.tex Describes the model class.

modelio.tex Describes the modelIO class.

objhandler.tex Describes the objHandler class.

popupwindows.tex Describes the popupWindows class.

surfwin.tex Describes the surfWin class.

trihandler.tex Describes the triHandler class.

triwin.tex Describes the triWin class.

Doc/Implementation/Figures

classOverview.eps The class overview figure (eps format).

Doc/Implementation/Figures/Xfig

classOverview.fig The class overview figure (xfig format).

Doc/Theory

Doc/Usersguide

general.tex Overview of Surfan.

userinterface.tex Describes how to use Surfan.

Doc/Usersguide/Figures

mainwindow.ps Main window overview (eps format).

modelandgraphwindow.ps Model and Graph Window overview (eps format).

progview.eps Program overview (eps format).

triangulation.ps Triangulation Window overview (eps format).

Doc/Usersguide/Figures/Xfig

progview.fig Program overview (xfig format).

Lib

README Information about this directory.

Obj

README Information about this directory.

Src

README Information about this directory.

mainwin.cpp Main window class file.

mainwin.h Main window class header file.

makefile Make information for Surfan.

model.cpp Model class file.

model.h Model class header file.

surfan.cpp The main file for Surfan.

surfwin.cpp Surfwin class file.

surfwin.h Surfwin class header file.

triwin.cpp Triwin class file.

triwin.h Triwin class header file.

Src/Include

color.cpp Color class file.

color.h Color class header file.

dispwin.cpp Dispwin class file.

dispwin.h Dispwin class header file.

graphHandler.cpp GraphHandler class file.

graphHandler.h GraphHandler class header file.

makefile Make the object files of all source files in this directory.

menu.cpp Menu class file.

menu.h Menu class header file.

popupwindows.cpp Popupwindows class file.

popupwindows.h Popupwindows class header file.

trackball.c Trackball (3D navigation) file.

trackball.h Trackball header file.

triHandler.h Trihandler class header file.

ui.c Used with the Trackball code.

ui.h Used with the Trackball code.

vect.c Used with the Trackball code.

vect.h Used with the Trackball code.

Src/Include/LEDA

Notes on the file structure

Before compiling, the above files are the only ones that exist in the surfan directories. Running 'make' in the Src directory will create all necessary object files that are put into the Obj directory and an executable file 'surfan' in the Bin directory. Running 'make' in the Doc directory creates this documentation.

Chapter 8

Future enhancements

8.1 Connectivity Graph

Automatic construction of a connectivity graph will be implemented. After this graph is calculated, the user may edit it to correst minor errors that may have resulted.

8.2 Graph Theory

Certain graph theoretical operations should be implemented, such as planarity testing, embedding of the graph, spanning trees, flows, k-connected components, decompositions etc.

8.3 Triangulation

An efficient triangulation, first of a surface and later of a volume should be made.

8.4 Image Processing

An image can be regarded as a weighted graph, where the weights represent the color coding of the image. Discrete differential geometry and spectral methods can then be used for edge detection, shape recognition, 3D reconstruction etc.

Bibliography

[1] Atiyah *Collected Works*

[2] Battista, Eades, Tamassia, Tolis *Algorithms for Drawing Graphs: an annotated bibliography* (1994)

[3] Beurling, Deny *Espaces de Dirichlet* (1950's)

[4] Biggs *Algebraic Graph Theory* (1993)

[5] Bonnington, Little *The Foundations of Topological Graph Theory* (1995)

[6] Cho, Park *Linear Connections on Graphs* (1996)

[7] Chung *Spectral Graph Theory* (1996)

[8] Connes *Non-Commutative Geometry* (1994)

[9] Dahlberg *Differential Geometry of Curves in R^n* (1996)

[10] Diestel *Graph Decompositions* (1989)

[11] Dimakis, Muller-Hoissen *Discrete Differential Calculus, Graphs, Topologies and Gauge Theory* (1994)

[12] Esposito *Dirac Operator and Spectral Geometry* (1997)

[13] A. Fomenko *Visual Geometry and Topology* Springer-Verlag 1994

[14] Gross, Tucker *Topological Graph Theory* (1987)

[15] D. Heller *Motif Programming Manual, version 1.1* O'Reilly & Associates Inc. 1991

[16] M. Himsolt *GML: Graph Modelling Language* (1996) URL: http://www.uni-passau.de/Graphlet/GML/

[17] Hopcraft, Tarjan *Efficient Planarity Testing* (1974)

[18] Max Planck Institute for Computer Science *LEDA - Library of Efficient Datatypes and Algorithms* URL: http://www.mpi-sb.mpg.de/LEDA/leda.html

[19] Nakahara *Geometry Topology and Physics* (1995)

[20] Narayanan, Vranas *A Numerical Test of the Continuum Index Theorem on the Lattice* (1997)

[21] OpenGL Architecture Review Board *OpenGL Programming Guide, second edition*, Addison Wesley, 1997

[22] L. Piegl, W. Tiller *The NURBS Book* Springer-Verlag 1995

[23] Rosenberg *The Laplacian on a Riemannian Manifold* (1997)

[24] B. Stroustrup *The C++ Programming Language, second edition* Addison-Wesley 1994

[25] D. Taylor *DTNURBS* URL: http://dtnet33-199.dt.navy.mil/dtnurbs/dtnurbs.htm

[26] D. Young *Object-Oriented Programming with C++ and OSF/Motif* Prentice Hall 1992

Index

Part IV

Appendices

Appendix A

IGES Curve and Surface Entity Types

A.1 Overview

Name	Type number	Section
Circular Arc	100	Curves
Composite Curve	102	Curves
Conic Arc	104	Curves
Copious Data	106	Special
Plane	108	Surfaces
Line	110	Curves
Parametric Spline Curve	112	Curves
Parametric Spline Surface	114	Surfaces
Point	116	Special
Ruled Surface	118	Surfaces
Surface of Revolution	120	Surfaces
Tabulated Cylinder	122	Surfaces
Transformation Matrix	124	Special
Flash	125	Special
Rational B-Spline Curve	126	Curves
Rational B-Spline Surface	128	Surfaces
Offset Curve	130	Special
Offset Surface	140	Special
Boundary	141	Special
Curve on a parametric surface	142	Curves
Bounded Surface	143	Surfaces
Trimmed (parametric) surface	144	Surfaces

A.2 Curves

Circular Arc, type 100

Index	Name	Type	Description
1	ZT	Real	Parallell z displacement of arc from XT,YT plane
2	X1	Real	Arc center abscissa
3	Y1	Real	Arc center ordinate
4	X2	Real	Start point abscissa
5	Y2	Real	Start point ordinate
6	X3	Real	End point abscissa
7	Y3	Real	End point ordinate

Composite Curve, type 102

Index	Name	Type	Description
1	N	Integer	Number of entities
2	DE1	Pointer	Pointer to the DE of the first constituent entry
⋮	⋮	⋮	
1+N	DEN	Pointer	Pointer to the DE of the last constituent entry

Conic Arc, type 104

Form number gives the type of arc; ellipse, hyperbola or parabola

Index	Name	Type	Description
1	A	Real	Conic Coefficient
2	B	Real	Conic Coefficient
3	C	Real	Conic Coefficient
4	D	Real	Conic Coefficient
5	E	Real	Conic Coefficient
6	F	Real	Conic Coefficient
7	ZT	Real	Parallell z displacement of arc from XT,YT plane
8	X1	Real	Start point abscissa
9	Y1	Real	Start point ordinate
10	X2	Real	End point abscissa
11	Y2	Real	End point ordinate

Line, type 110

Form number tells whether the plane is bounded or unbounded

Index	Name	Type	Description
1	X1	Real	Start point (P1) x-coordinate
2	Y1	Real	Start point (P1) y-coordinate
3	Z1	Real	Start point (P1) z-coordinate
4	X2	Real	Start point (P2) x-coordinate
5	Y2	Real	Start point (P2) y-coordinate
6	Z2	Real	Start point (P2) z-coordinate

Parametric Spline Curve, type 112

Index	Name	Type	Description
1	CTYPE	Integer	Spline type:
			1 = Linear
			2 = Quadratic
			3 = Cubic
			4 = Wilson-Fowler
			5 = Modified Wilson-Fowler
			6 = B-spline
2	H	Integer	Degree of continuity with respect to arc length
3	NDIM	Integer	Number of dimensions:
			2 = planar
			3 = non-planar
4	N	Integer	Number of segments
5	T(1)	Real	First breakpoint of piecewise polynomial
⋮	⋮	⋮	
5+N	T(N+1)	Real	Last break point of piecewise polynomial
6+N	AX(1)	Real	X coordinate polonomial
7+N	BX(1)	Real	
8+N	CX(1)	Real	
9+N	DX(1)	Real	
10+N	AY(1)	Real	Y coordinate polonomial
11+N	BY(1)	Real	
12+N	CY(1)	Real	
13+N	DY(1)	Real	
14+N	AZ(1)	Real	Z coordinate polonomial
15+N	BZ(1)	Real	
16+N	CZ(1)	Real	
17+N	DZ(1)	Real	
⋮	⋮	⋮	Subsequent X, Y, Z polynomials concluding with the twelve coefficients of the Nth polynomial segment.

The parameters that follow comprise the evaluations of the polynomials of the N-th segment and their derivatives at the parameter value u=T(N+1) corresponding to the terminate point. Subsequently, these evaluations are divided by the appropriate factorials.

6+13N	TPX0	Real	X value
7+13N	TPX1	Real	X first derivative
8+13N	TPX2	Real	X second derivative / 2!
9+13N	TPX3	Real	X third derivative / 3!
10+13N	TPY0	Real	Y value
11+13N	TPY1	Real	Y first derivative
12+13N	TPY2	Real	Y second derivative / 2!
13+13N	TPY3	Real	Y third derivative / 3!
14+13N	TPZ0	Real	Z value
15+13N	TPZ1	Real	Z first derivative
16+13N	TPZ2	Real	Z second derivative / 2!
17+13N	TPZ3	Real	Z third derivative / 3!

Rational B-Spline Curve, type 126

Form number represents a preferred curve type, see 4.20

Index	Name	Type	Description
1	K	Integer	Upper index of sum
2	M	Integer	Degree of basis functions
3	PROP1	Integer	0 = nonplanar, 1 = planar
4	PROP2	Integer	0 = open curve, 1 = closed curve
5	PROP3	Integer	0 = rational, 1 = polynomial
6	PROP4	Integer	0 = nonperiodic, 1 = periodic

Let W = 1+K-M and A = N+2M

7	T(-M)	Real	First value of knot sequence
⋮	⋮	⋮	
7+A	T(N+M)	Real	Last value of knot sequence
8+A	W(0)	Real	First weight
⋮	⋮	⋮	
8+A+K	W(K)	Real	Last weight
9+A+K	X0	Real	First control point
10+A+K	Y0	Real	
11+A+K	Z0	Real	
⋮	⋮	⋮	
9+A+4K	XK	Real	Last control point
10+A+4K	YK	Real	
11+A+4K	ZK	Real	
12+A+4K	V(0)	Real	Starting parameter value
13+A+4K	V(1)	Real	Ending parameter value
14+A+4K	XNORM	Real	Unit normal (if curve is planar)
15+A+4K	YNORM	Real	
16+A+4K	ZNORM	Real	

Curve on a parametric surface, type 142

Form number represents a preferred curve type, see 4.20

Index	Name	Type	Description
1	CRTN	Integer	Indicates the way the curve on the surface has been created: 0 = Unspecified 1 = Projection of a given curve on the surface 2 = Intersection of two surfaces 3 = Isoparametric curve, i.e. either a u-parametric or a v-parametric curve.
2	SPTR	Pointer	Pointer to the DE of the surface on which the curve lies.
3	BPTR	Pointer	Pointer to the DE of the entity that contains the definition of the curve B in the parametric space (u,v) of the surface S.
4	CPTR	Pointer	Pointer to the DE of the curve C
5	PREF	Integer	Indicates preferred representation in the sending system: 0 = Unspecified 1 = S o B is preferred 2 = C is preferred 3 = C and S o B are equally preferred

A.3 Surfaces

Plane, type 108

Form number tells whether the plane is bounded or unbounded

Index	Name	Type	Description
1	A	Real	Coefficients of plane
2	B	Real	Coefficients of plane
3	C	Real	Coefficients of plane
4	D	Real	Coefficients of plane
5	PTR	Pointer	Pointer to the DE of the closed curve entity (Zero for unbounded plane)
6	X	Real	XT coordinate of location point for display symbol
7	Y	Real	YT coordinate of location point for display symbol
8	Z	Real	ZT coordinate of location point for display symbol
9	SIZE	Real	Size parameter for display symbol

Parametric Spline Surface, type 114

Index	Name	Type	Description
1	CTYPE	Integer	Spline Boundary Type:
			1 = Linear
			2 = Quadratic
			3 = Cubic
			4 = Wilson-Fowler
			5 = Modified Wilson-Fowler
			6 = B-spline
2	PTYPE	Integer	Patch Type
			1 = Cartesian Product
			0 = Unspecified
3	M	Integer	Number of u segments
4	N	Integer	Number of v segments
5	TU(1)	Real	First breakpoint in u (u values of grid lines)
⋮	⋮	⋮	
5+M	TU(M+1)	Real	Last break point in u
6+M	TV(1)	Real	First breakpoint in v (v values of grid lines)
⋮	⋮	⋮	
6+M+N	TV(N+1)	Real	Last break point in v
7+M+N	AX(1,1)	Real	First X coefficient of (1,1) patch
⋮	⋮	⋮	
22+M+N	SX(1,1)	Real	Last X coefficient of (1,1) patch
23+M+N	AY(1,1)	Real	First Y coefficient of (1,1) patch
⋮	⋮	⋮	
38+M+N	SY(1,1)	Real	Last Y coefficient of (1,1) patch

Index	Name	Type	Description
39+M+N	AZ(1,1)	Real	First Z coefficient of (1,1) patch
⋮	⋮	⋮	
54+M+N	SZ(1,1)	Real	Last Z coefficient of (1,1) patch
55+M+N	AX(1,2)	Real	First X coefficient of (1,2) patch
⋮	⋮	⋮	
102+M+N	SZ(1,2)	Real	Last Z coefficient of (1,2) patch
⋮	⋮	⋮	
7+M+N+48(N-1)	AX(1,N)	Real	First X coefficient of (1,N) patch
⋮	⋮	⋮	
6+M+N+48N	SZ(1,N)	Real	Last Z coefficient of (1,N) patch
7+M+N+48N	¡n.a.¿	Real	Beginning of arbitrary value
⋮	⋮	⋮	
6+M+N+48(N+1)	¡n.a.¿	Real	End of arbitrary value
7+M+N+48(N+1)	AX(2,1)	Real	First X coefficient of (2,1) patch
⋮	⋮	⋮	
6+M+N+48(N+6)	SZ(2,1)	Real	Last Z coefficient of (2,1) patch
⋮	⋮	⋮	
7+M+N+48(2N)	AX(2,N)	Real	First X coefficient of (2,N) patch
⋮	⋮	⋮	
6+M+N+48(2N+1)	SZ(2,N)	Real	Last Z coefficient of (2,N) patch
7+M+N+48(2N+1)	¡n.a.¿	Real	Beginning of arbitrary value
⋮	⋮	⋮	
6+M+N+48(2N+2)	¡n.a.¿	Real	Arbitrary value
⋮	⋮	⋮	
7+M+N+48[(J-1)(N+1)+K-1]	AX(J,K)	Real	First X coefficient of (J,K) patch
⋮	⋮	⋮	
6+M+N+48[(J-1)(N+1)+K]	SZ(J,K)	Real	Last Z coefficient of (J,K) patch
⋮	⋮	⋮	
7+M+N+48[(M-1)(N+1)+N-1]	AX(M,N)	Real	First X coefficient of (M,N) patch
⋮	⋮	⋮	
6+M+N+48[(M-1)(N+1)+N]	SZ(M,N)	Real	Last Z coefficient of (M,N) patch
7+M+N+48[(M-1)(N+1)+N]	¡n.a.¿	Real	Beginning of arbitrary value
⋮	⋮	⋮	
6+M+N+48[(M-1)(N+1)+N+1]	¡n.a.¿	Real	Arbitrary value
7+M+N+48[M(N+1)]	¡n.a.¿	Real	Arbitrary value
⋮	⋮	⋮	
6+M+N+48[M(N+1)+N+1]	¡n.a.¿	Real	End of arbitrary value

Ruled Surface, type 118

Index	Name	Type	Description
1	DE1	Pointer	Pointer to the DE of the first curve entity
2	DE2	Pointer	Pointer to the DE of the second curve entity
3	DIRFLG	Integer	Direction flag:
			0 = Join first to first, last to last
			1 = Join first to last, last to first
4	DEVFLG	Integer	Developable surface flag:
			1 = Developable
			2 = Possibly not

Surface of Revolution, type 120

Index	Name	Type	Description
1	L	Pointer	Pointer to the DE of the Line entity (axis of revolution)
2	C	Pointer	Pointer to the DE of the generatrix entity
3	SA	Real	Start angle in radians
4	TA	Real	Terminate angle in radians

Tabulated Cylinder, type 122

Index	Name	Type	Description
1	DE	Pointer	Pointer to the DE of the Directrix curve entity
2	LX	Real	Coordinate of the terminate point of the generatrix.
3	LY	Real	The start point of the generatrix is identical with
4	LZ	Real	the start point of the directrix.

Rational B-Spline Surface, type 128

Form number represents a preferred surface type, see 4.21

Index	Name	Type	Description
1	K1	Integer	Upper index of first sum
2	K2	Integer	Upper index of second sum
3	M1	Integer	Degree of first set of basis functions
4	M2	Integer	Degree of second set of basis functions
5	PROP1	Integer	1 = Closed in first parametric variable direction
			0 = Not closed
6	PROP2	Integer	1 = Closed in second parametric variable direction
			0 = Not closed
7	PROP3	Integer	0 = Rational, 1 = Polynomial
8	PROP4	Integer	0 = nonperiodic in first parametric variable direction
			1 = periodic in first parametric variable direction
9	PROP5	Integer	0 = nonperiodic in second parametric variable direction
			1 = periodic in second parametric variable direction

Let N1 = 1+K1-M1
N2 = 1+K2-M2
A = N1+2*M1
B = N2+2*M2
C = (1+K1)(1+K2)

10	S(-M1)	Real	First value of first knot sequence
⋮	⋮	⋮	
10+A	S(N1+M1)	Real	Last value of first knot sequence
11+A	T(-M2)	Real	First value of second knot sequence
⋮	⋮	⋮	
11+A+B	T(N2+M2)	Real	Last value of second knot sequence
12+A+B	W(0,0)	Real	First weight
13+A+B	W(1,0)	Real	
⋮	⋮	⋮	
11+A+B+C	W(K1,K2)	Real	Last weight
12+A+B+C	X(0,0)	Real	First control point
13+A+B+C	Y(0,0)	Real	
14+A+B+C	Z(0,0)	Real	
15+A+B+C	X(1,0)	Real	
16+A+B+C	Y(1,0)	Real	
17+A+B+C	Z(1,0)	Real	
⋮	⋮	⋮	
9+A+B+4C	X(K1,K2)	Real	Last control point
10+A+B+4C	Y(K1,K2)	Real	
11+A+B+4C	Z(K1,K2)	Real	
12+A+B+4C	U(0)	Real	Starting value for first parametric direction
13+A+B+4C	U(1)	Real	Ending value for first parametric direction
14+A+B+4C	V(0)	Real	Starting value for second parametric direction
15+A+B+4C	V(1)	Real	Ending value for second parametric direction

Trimmed (parametric) surface, type 144

Index	Name	Type	Description
1	PTS	Pointer	Pointer to the DE of the Surface entity that is to be trimmed
2	N1	Integer	0 = The outer boundary is the boundary of D 1 = otherwise
3	N2	Integer	This number indicates the number of simple closed curves which constitute the inner boundary of the trimmed surface. In case no inner boundary is introduced, this is set equal to zero.
4	PT0	Pointer	Pointer to the DE of the simple closed curve entity (Curve on a Parametric Surface Entity), that constitutes the outer boundary of the trimmed surface or zero.
5	PTI1	Pointer	Pointer to the DE of the first simple closed inner boundary curve entity (Curve on a Parametric Surface Entity), according to some arbitrary ordering of these entities.
$\vdots$	$\vdots$	$\vdots$	
4+N2	PTIN2	Pointer	Pointer to the DE of the last simple closed inner boundary curve entity (Curve on a Parametric Surface Entity).

A.4 Special

Copious Data, type 106

Form number gives the data point storage format; see 4.6

Index	Name	Type	Description
1	IP	Integer	Interpretation flag
			1 = x,y pairs, common z
			2 = x,y,z coordinates
			3 = x,y,z coordinates and i,j,k vectors
2	N	Integer	Number of n-tuples

For IP=1 (x,y pairs, common z), i.e. for form 1

Index	Name	Type	Description
3	ZT	Real	Common z displacement
4	X1	Real	First data point abscissa
5	Y1	Real	First data point ordinate
3+2N	YN	Real	Last data point ordinate

For IP=2 (x,y,z triples), i.e. for form 2

Index	Name	Type	Description
3	X1	Real	First data point x value
4	Y1	Real	First data point y value
5	Z1	Real	First data point z value
⋮	⋮	⋮	
2+3N	ZN	Real	Last data point z value

For IP=3 (x,y,z,i,j,k sextuples), i.e. for form 3

Index	Name	Type	Description
3	X1	Real	First data point x value
4	Y1	Real	First data point y value
5	Z1	Real	First data point z value
6	I1	Real	First data point i value
7	J1	Real	First data point j value
8	K1	Real	First data point k value
⋮	⋮	⋮	
2+6N	KN	Real	Last data point k value

Point, type 116

Index	Name	Type	Description
1	X	Real	X Coordinate of point
2	Y	Real	Y Coordinate of point
3	Z	Real	Z Coordinate of point
4	PTR	Pointer	Pointer to the DD of the subfigure definition entity specifying the display symbol or zero. If zero, no display symbol specified.

Transformation Matrix, type 124

Index	Name	Type	Description
1	R11	Real	Top row
2	R12	Real	
3	R13	Real	
4	T1	Real	
5	R21	Real	Second row
6	R22	Real	
7	R23	Real	
8	T2	Real	
9	R31	Real	Third row
10	R32	Real	
11	R33	Real	
12	T3	Real	

Flash, type 125

Index	Name	Type	Description
1	X	Real	X reference of flash
2	Y	Real	Y reference of flash
3	DIM1	Real	First flash sizing parameter
4	DIM2	Real	Second flash sizing parameter
5	ROT	Real	Rotation of flash about reference point in radians
6	DE	Pointer	Pointer to the DE of the referenced entity or zero

Offset Curve, type 130

Index	Name	Type	Description
1	DE1	Pointer	Pointer to the DE of the curve entity to be offset
2	FLAG	Integer	Offset distance flag:
			1 = Single value offset, uniform distance
			2 = Offset distance varying linearly
			3 = Offset distance as a specified function
3	DE2	Pointer	Pointer to the DE of the curve entity, one coordinate
			of which describes the offset as a function
			of its parameter
		or 0	(unless FLAG = 3)
4	NDIM	Integer	Pointer of particular coordinate of DE2 which describes
			offset as a function of its parameter
			(only used if FLAG = 3)
5	PTYPE	Integer	Tapered offset type flag:
			1 = Function of arc length
			2 = Function of parameter
			(only used if FLAG = 2 or 3)
6	D1	Real	First offset distance (only used if FLAG = 1 or 2)
7	TD1	Real	Arc length of parameter value, depending on PTYPE,
			of first offset distance. (only used if FLAG = 2)
8	D2	Real	Second offset distance
9	TD2	Real	Arc length of parameter value, depending on PTYPE,
			of second offset distance. (only used if FLAG = 2)
10	VX	Real	X-component of unit vector normal to plane containing
			curve to be offset
11	VY	Real	Y-component of unit vector normal to plane containing
			curve to be offset
12	VZ	Real	Z-component of unit vector normal to plane containing
			curve to be offset
13	TT1	Real	Offset curve starting parameter value
14	TT2	Real	Offset curve ending parameter value

Offset Surface, type 140

Index	Name	Type	Description
1	NX	Real	The X-coordinate of the offset indicator N(Um,Vm)
2	NY	Real	The Y-coordinate of the offset indicator N(Um,Vm)
3	NZ	Real	The Z-coordinate of the offset indicator N(Um,Vm)
4	D	Real	The distance by which the surface is normally offset
			on the side of the offset indicator if d¿0
			and on the opposite side if d¡0.
5	DE	Pointer	Pointer to the DE of the surface entity to be offset

Boundary, type 141

This entity is untested!

Index	Name	Type
1	TYPE	Integer
2	PREF	Integer
3	SPTR	Pointer
4	N	Integer
5	CRVPT1	Pointer
6	SENSE1	Integer
7	K1	Integer
8	PSCPT1	Pointer
⋮	⋮	⋮
7+K1	PSCPTK1	Pointer
⋮	⋮	⋮

Let M=5+3*(N-1)+(K1+K2+ ...+KN-1)

M	CRVPTN	Pointer
1+M	SENSEN	Integer
2+M	KN	Integer
3+M	PSCPTN	Pointer
⋮	⋮	⋮
KN+M	PSCPTKN	Pointer

Bounded surface, type 143

This entity is untested!

Index	Name	Type
1	TYPE	Integer
2	SPTR	Pointer
3	N	Integer
4	BDPT1	Pointer
⋮	⋮	⋮
3+N	BDPTN	Pointer

A.5 Discarded information about an IGES object

Number	Field name
2	Parameter data (the pointer)
3	Structure
4	Line font pattern
5	Level
6	View
8	Label Display Associativity
9	Status number
10	Sequence number
11	Entity Type number (again)
12	Line Weight number
13	Color number
16	(Reserved for future use)
17	(Reserved for future use)
18	Entity label
19	Entity Subscript number
10	Sequence number