P. A. Luchshev, N. G. Gulub

COMPUTER GRAPHICS

2024

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
National Aerospace University
«Kharkiv Aviation Institute»

P. A. Luchshev, N. G. Gulub

COMPUTER GRAPHICS

Tutorial

Kharkiv «KhAI» 2024

UDC 004.92
L87

Розглянуто принципи та методи створення програмних додатків із застосуванням комп'ютерної графіки за допомогою бібліотеки OpenGL, сучасних мов програмування (C#, C++) і об'єктно-орієнтованої моделі програмування.

Для студентів напряму «Інженерія програмного забезпечення» при виконанні практичних робіт, повторенні й вивченні принципів ООП.

Previewers: Doctor of Science, Professor M. Volk,
Condidate of Science, Associate Professor T. Filimonchuk

Luchshev, P. A.
L87 Computer graphics [Electronic resource] : tutorial / P. A. Luchshev, N. G. Gulub. – Kharkiv: National Aerospace University «Kharkiv Aviation Institute», 2024. – 80 c.

The principles and methods of creating software applications that use OpenGL computer graphics, modern programming languages (C#, C++) and an object-oriented programming model are considered.

For students of the specialty «Software Engineering» for the implementation of practical work on 2D and 3D computer graphics, game development, studying and deepening knowledge of OOP.

Figures 23. Tables 14. Bibliogr. : 12 titles.

UDC 004.92

INTRODUCTION

The C# language, the Visual Studio integrated development environment and the OpenGL library are used as the basic tools for the practical work. Usage of other programming languages, for example, C++, Java and the required software development tools is allowed in agreement with the teacher.

Each practical work is planned to be completed in two weeks. The execution of practical work means the development of a software application in according to the variant of the task and the preparation of a final report. After that the file with report and the *zip*-archive containing developed software project are sent to the student work registration system specified by the teacher, for example: *mentor.khai.edu*, *Google Classroom* etc. It should be noted that student work registration systems record the completion time of student projects and subsequent archiving of all practical work. To pass practical work and to be assessed, the student must personally present his work to the teacher.

Typically a report includes the following parts:

– title page;
– personal task according to variant;
– theoretical information about the graphics solutions are used;
– listing of the program with the task implementation;
– one or several screenshorts of the running program;
– the assessment table with marks of objectives completion.

The archive should contain one directory with the source code of the software project. That's enough to recompile the project. It's required that the names of the archive and report file are the same, and only their extensions differ. Also the file name should contain the group number, student surname, practical work number, for example:

«631 Petrov #3.docx» – report file;
«631 Petrov #3.zip» – project's files zip-archive.

You can use e-mail *mailto:computer.graphix@gmail.com* to discuss organizational issues.

Practical work № 1

THE PRINCIPLES OF USING OPENGL, BASIC POSSIBILITIES AND COMMANDS

Aim of work: study the features of creating simple software applications that use the *OpenGL* library. Learn how to draw flat convex shapes using geometric primitives and the coordinate system setting by OpenGL commands.

Task

Using the teacher specified development tools, create a simple software project that supports the *OpenGL*. Considering the evaluation system (Table 1.1), develop a software application program using the *OpenGL* commands [1, 2, 6], which sets coordinate system creates and displays the image on the screen or window, according to specified primitives (Table 1.2) and coordinate limits $\boldsymbol{x_1, y_1}$ and $\boldsymbol{x_2, y_2}$ in the task variant. Dotted or dash or dash-dotted lines must be used to draw the grid. The contour of the figure must be drawn with a bold line (more than one pixel). For even variants, the points of figure should be square, and for odd ones, round.

Methodical instructions

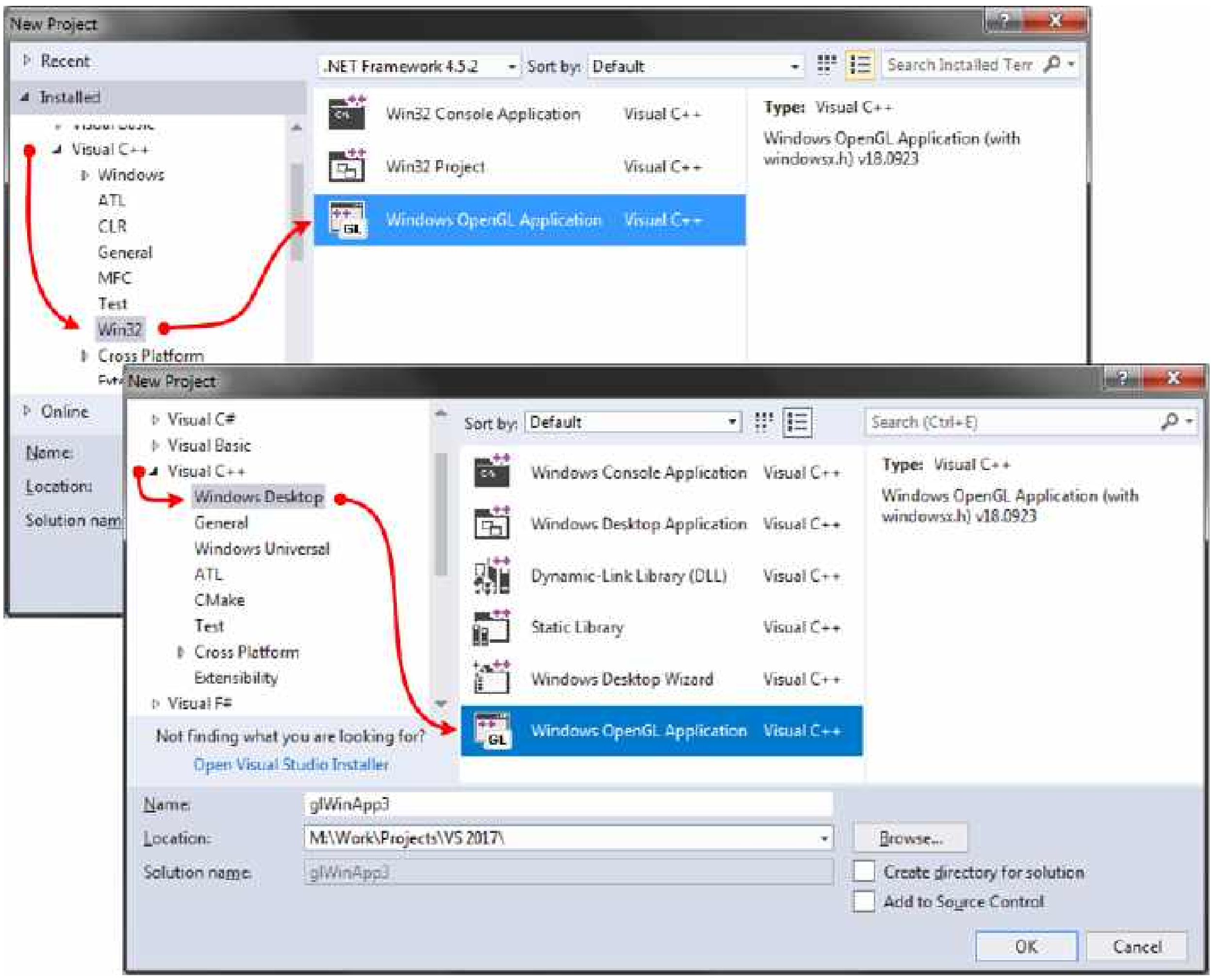
OpenGL graphic subsystem is supported on various operating systems and can be linked to many programming languages. The following is a step-by-step instruction to create of C++ and C# applications on the Windows platform after installing the projects templates (see Appendix 1).

OpenGL programming and getting started with C++

To create a new program, select ***Windows OpenGL Application*** Project Template. Note that the Project Template may be in different places depending on the versions of ***Visual Studio***. For example, in Fig. 1.1 is shown the location for ***Visual Studio 2015*** and ***Visual Studio 2017*** project template.

The project includes several source files. Pay attention to following files (Fig. 1.2):

- ***StdWindow.cpp*** – Implementation of standard activities to initialization, window creation and organization of a message processing loop for *Windows* operating system events. This source code have to be changed if you add standard operating system controls to the application to organize the user interface (menus, dialog boxes, etc.).
- ***glWinApp.cpp*** – the location of the event's handlers and the applied *OpenGL* code. It is assumed that the main part of the objectives will be implemented in this file (see Appendix 2 «Using the *windowsx.h* file in C++ projects») [10].



Type: Visual C++

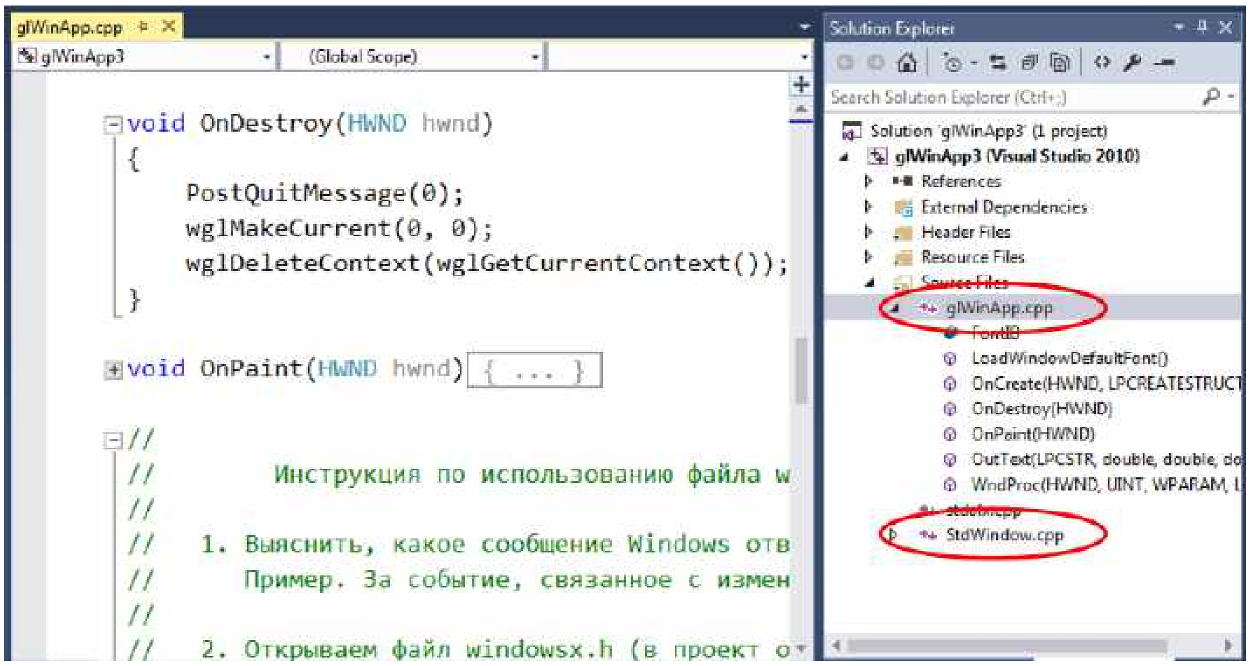
Windows OpenGL Application (with windowsx.h) v18.0923

Name: glWinApp3

Location: M:\Work\Projects\VS 2017\

Solution name: glWinApp3

Fig. 1.1. OpenGL C++ Project Template location
in *Visual Studio* 2015 and 2017

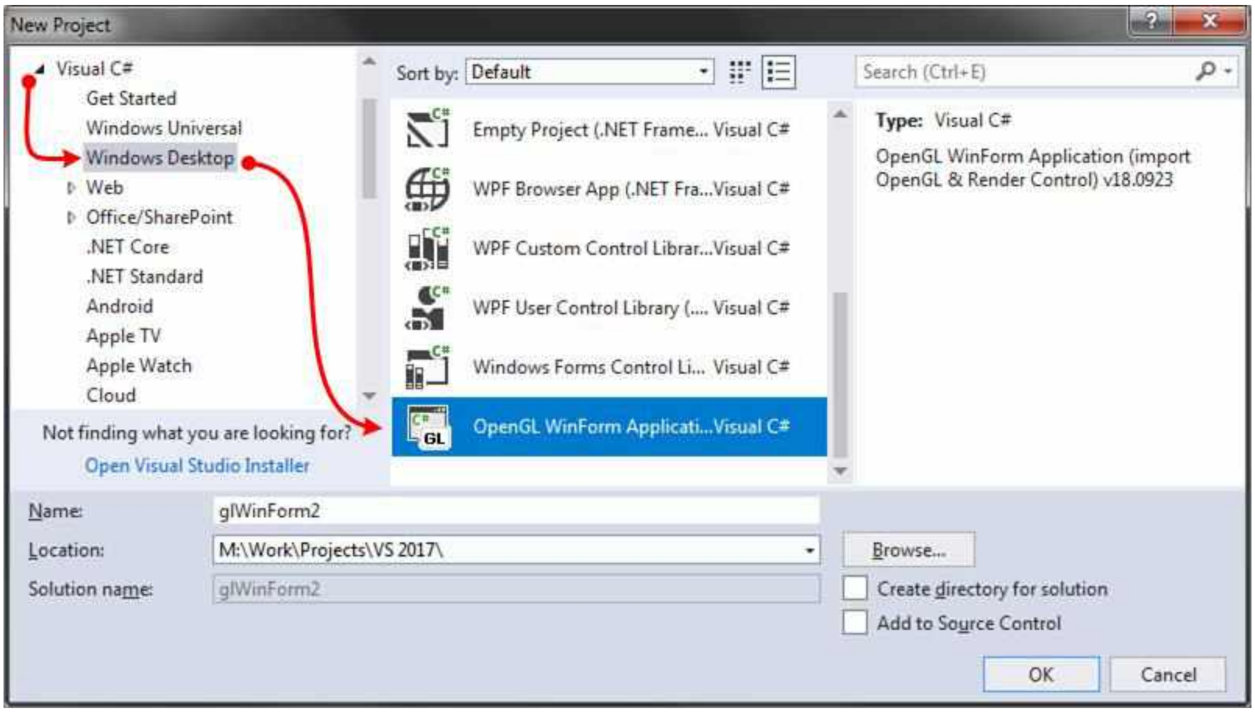


```
void OnDestroy(HWND hwnd)
{
    PostQuitMessage(0);
    wglMakeCurrent(0, 0);
    wglDeleteContext(wglGetCurrentContext());
}

void OnPaint(HWND hwnd){ ... }

//
//        Инструкция по использованию файла w
//
//    1. Выяснить, какое сообщение Windows отв
//       Пример. За событие, связанное с измен
//
//    2. Открываем файл windowsx.h (в проект о
```

Fig. 1.2. C++ Project Template files with *OpenGL*

OpenGL programming and getting started with *C#*

To create a new project, use the ***New Project*** dialog. Note that the location of the Project Template may vary by version of ***Visual Studio***, like for C++ projects. For example, after the automatic installation of the Project Templates (see Appendix 1), the dialog for creating a new C # project using ***OpenGL*** in ***Visual Studio 2017*** is shown on Fig.1.3.


New Project
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
Office/SharePoint
.NET Core
.NET Standard
Android
Apple TV
Apple Watch
Cloud
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
Empty Project (.NET Frame... Visual C#
WPF Browser App (.NET Fra...Visual C#
WPF Custom Control Librar...Visual C#
WPF User Control Library (.... Visual C#
Windows Forms Control Li... Visual C#
OpenGL WinForm Applicati...Visual C#
Browse...
Create directory for solution
Add to Source Control
OK
Cancel

Type: Visual C#
OpenGL WinForm Application (import OpenGL & Render Control) v18.0923

Name: glWinForm2
Location: M:\Work\Projects\VS 2017\
Solution name: glWinForm2

Fig. 1.3. Main C++ project template files with OpenGL support

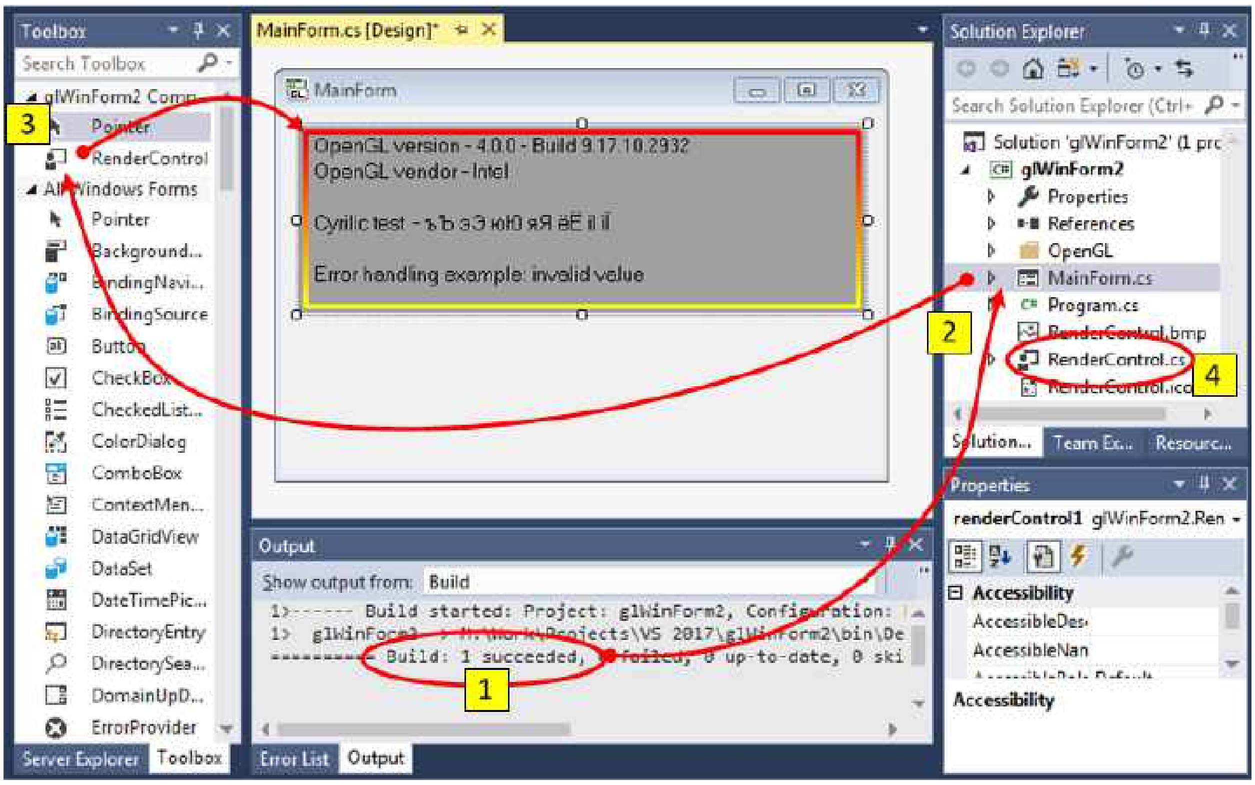
After creating the project, you need to compile the application (***Ctrl+Shift+B***), since it includes a component for working with OpenGL, which cannot be placed on the main form of the application without this step. Thus, after creating the project, the following set of actions is performed in the specified order (Fig. 1.4):

1. Compiling the project with ***Ctrl+Shift+B***.
2. Double-click to open the ***Main Form*** of application in designer mode.
3. Place the RenderControl (component for working with OpenGL) on the main application form.
4. Select the ***RenderControl.cs*** file in the project structure and press the ***F7*** key to proceed to editing the source code.

Checklist Questions

1. How do you get color value for emmited light?
2. How do you get color value for reflected light?
3. How is the point size set?
4. How is the line width set?
5. How is the line pattern (solid, dotted, etc.) set?
6. What commands are used to snap the coordinate system to the window size?

7. What is the difference between isotropic and anisotropic coordinate systems?



OpenGL version - 4.0.0 - Build 9.17.10.2932
OpenGL vendor - Intel

Cyrilic test - ъЪ эЭ юЮ яЯ ёЁ іІ

Error handling example: invalid value

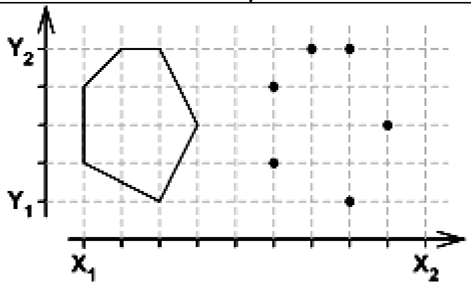
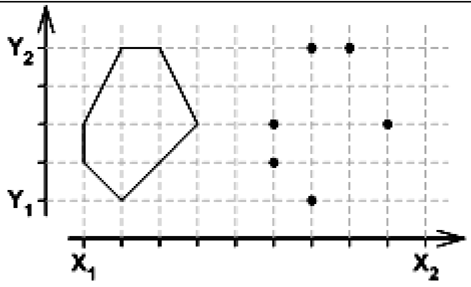
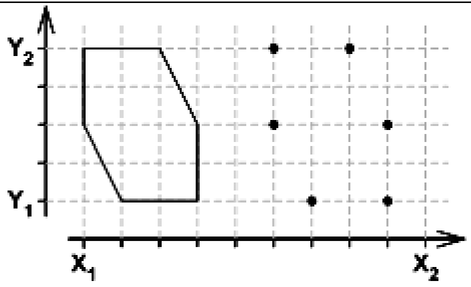
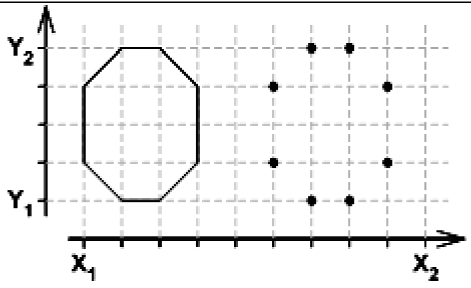
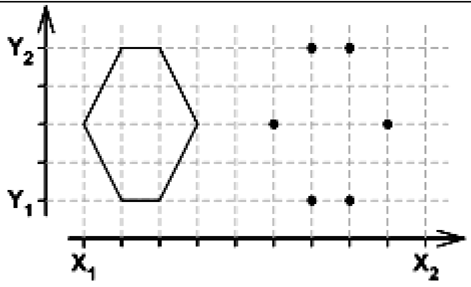
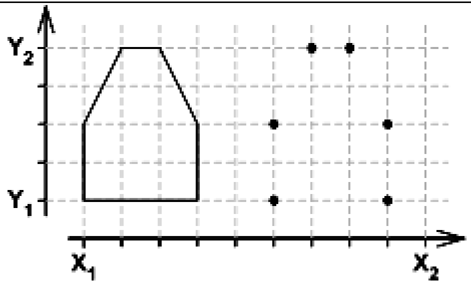
```
1>------ Build started: Project: glWinForm2, Configuration: 
1>  glWinForm2 -> H:\Work\Projects\VS 2017\glWinForm2\bin\De
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 ski
```

Fig. 1.4. Order of actions to start a C# application based on a project template with OpenGL support

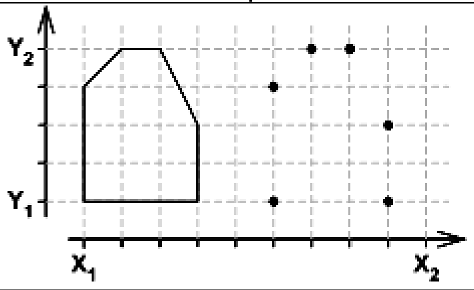
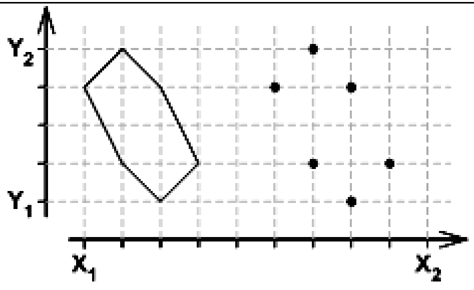
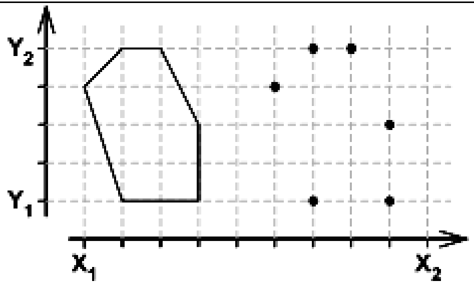
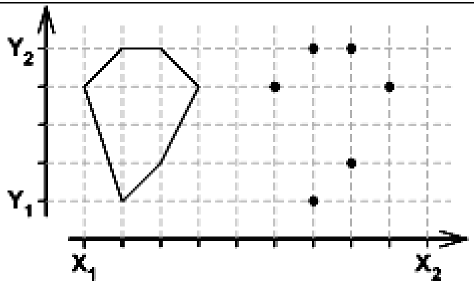
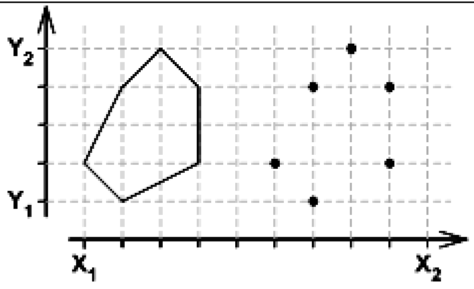
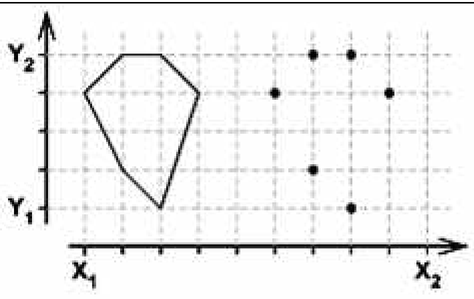
Table 1.1

No.	Complexity	Assignments	Points
1	Basic	When launching the application, the image matches the option	2
2		Correct display of the task when changing the size / position of the window	1
3		Development of routines to avoid code duplication	1
4		Using loops to create images	1
5	Advanced	Rendering the image with vector OpenGL commands (glDrawArrays, etc.)	1
6		Using OOP (developing your own classes)	2

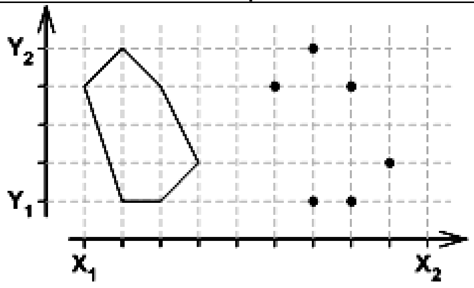
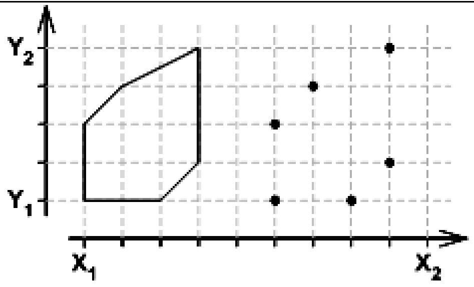
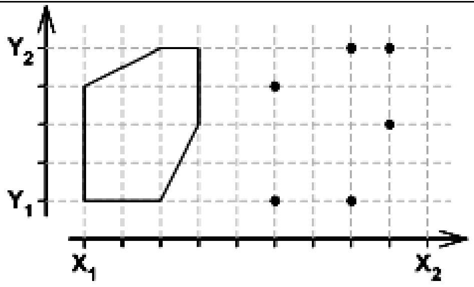
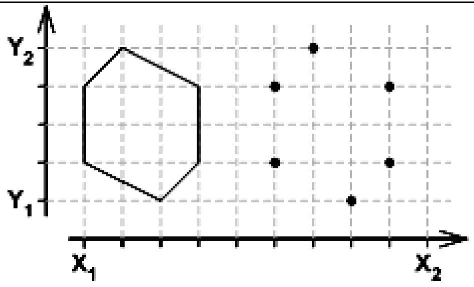
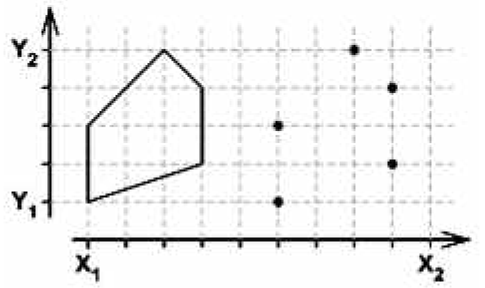
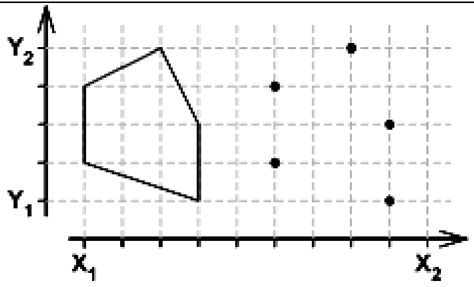
Table 1.2

No.	Parameters	Shape
1	Primitives: GL_POINTS, GL_LINES $x_1 = 0$; $x_2 = 9$ $y_1 = 0$; $y_2 = 4$	
2	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -1$; $x2 = 8$ $y_1 = -2$; $y2 = 2$	
3	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8$; $x_2 = 1$ $y_1 = -3$; $y_2 = 1$	
4	Primitives: GL_POINTS, GL_LINES $x_1 = -4$; $x_2 = 5$ $y_1 = -1$; $y_2 = 3$	
5	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -9$; $x_2 = 0$ $y_1 = -4$; $y_2 = 0$	
6	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -7$; $x_2 = 2$ $y_1 = -1$; $y_2 = 3$	

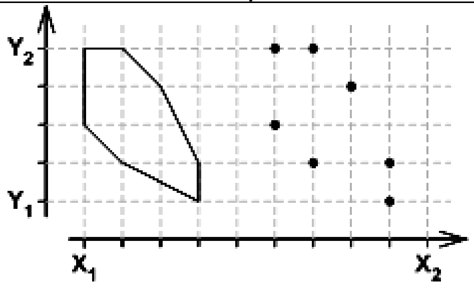
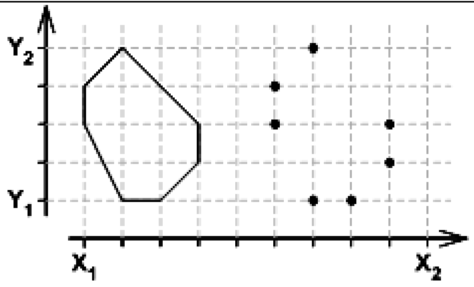
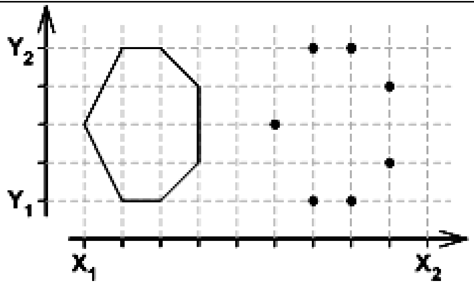
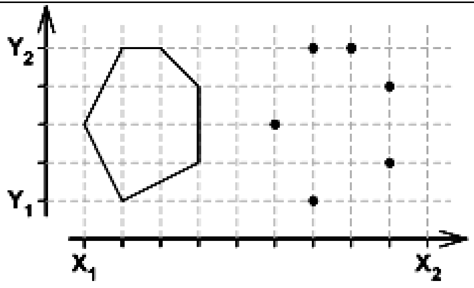
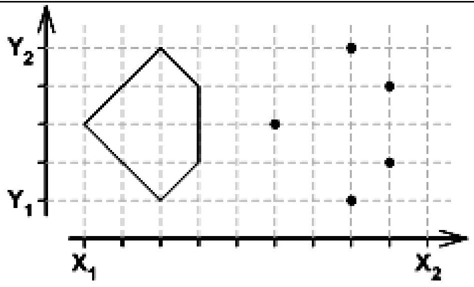
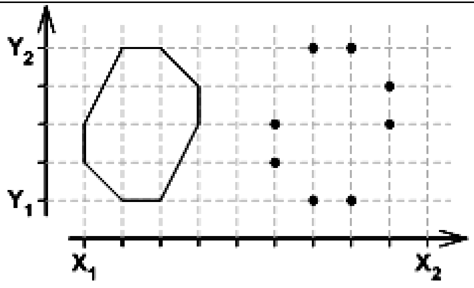
Continuation of Table 1.2

No.	Parameters	Shape
7	Primitives: GL_POINTS, GL_LINES $x_1 = 0$; $x_2 = 18$ $y_1 = 0$; $y_2 = 8$	
8	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2$; $x_2 = 16$ $y_1 = -4$; $y_2 = 4$	
9	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -16$; $x_2 = 2$ $y_1 = -6$; $y_2 = 2$	
10	Primitives: GL_POINTS, GL_LINES $x_1 = -8$; $x_2 = 10$ $y_1 = -2$; $y_2 = 6$	
11	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -18$; $x_2 = 0$ $y_1 = -8$; $y_2 = 0$	
12	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -14$; $x_2 = 4$ $y_1 = -2$; $y_2 = 6$	

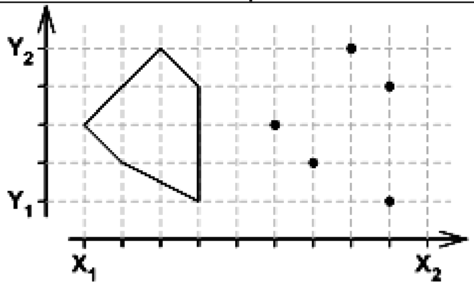
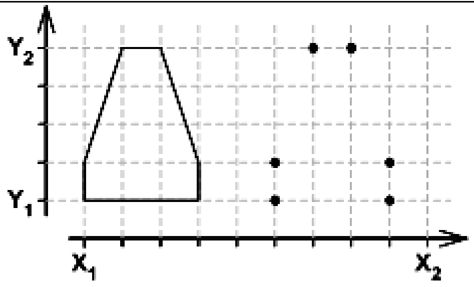
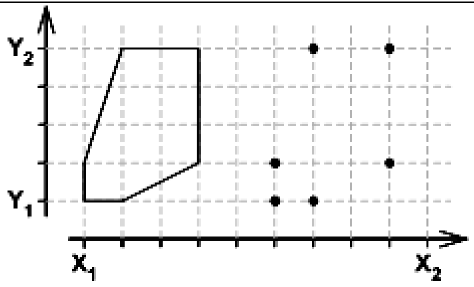
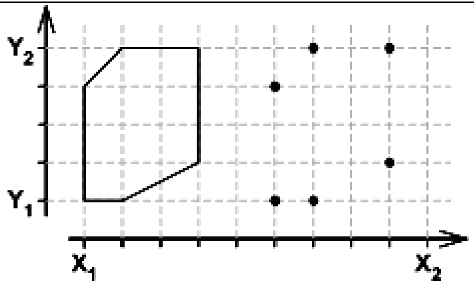
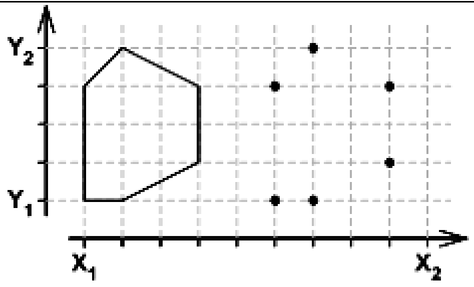
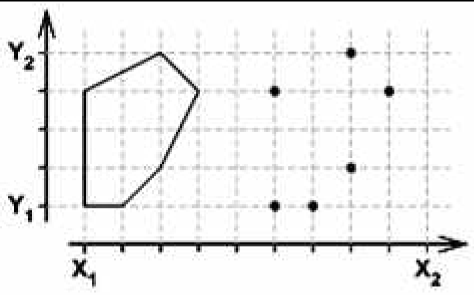
Continuation of Table 1.2

No.	Parameters	Shape
13	Primitives: GL_POINTS, GL_LINES: $x_1 = 0$; $x_2 = 4.5$ $y_1 = 0$; $y_2 = 2$	
14	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -0.5$; $x_2 = 4$ $y_1 = -1$; $y_2 = 1$	
15	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -4$; $x_2 = 0.5$ $y_1 = -1.5$; $y_2 = 0.5$	
16	Primitives: GL_POINTS, GL_LINES $x_1 = -2$; $x_2 = 2.5$ $y_1 = -0.5$; $y_2 = 1.5$	
17	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -4.5$; $x_2 = 0$ $y_1 = -2$; $y_2 = 0$	
18	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -3.5$; $x_2 = 1$ $y_1 = -0.5$; $y_2 = 1.5$	

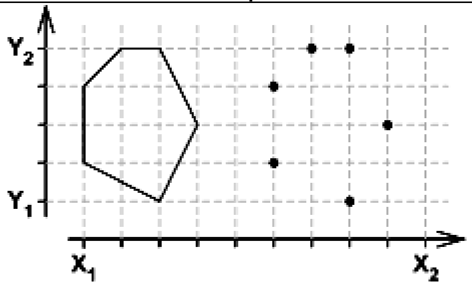
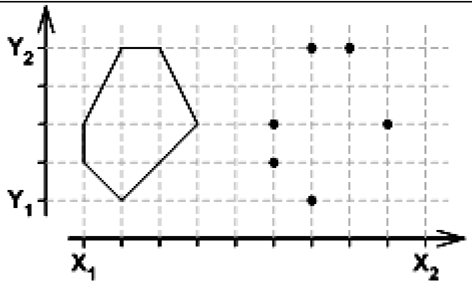
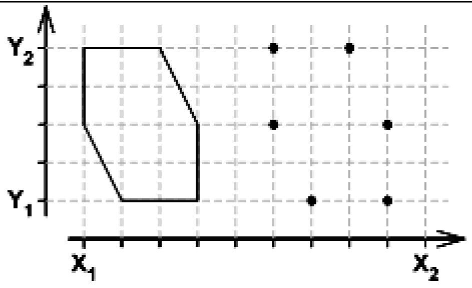
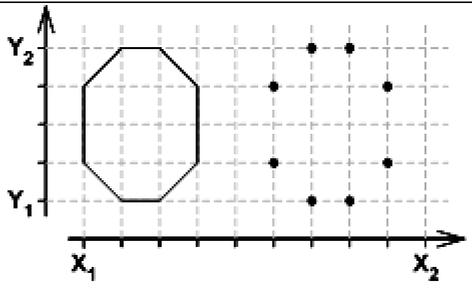
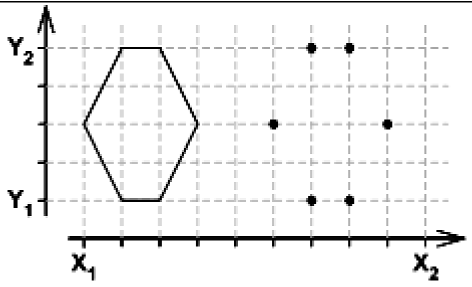
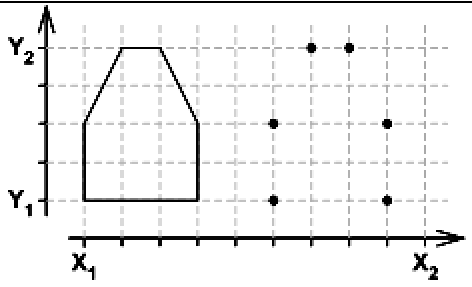
Continuation of Table 1.2

No.	Parameters	Shape
19	Primitives: GL_POINTS, GL_LINES $x_1 = -1$; $x_2 = 8$ $y_1 = -2$; $y_2 = 2$	
20	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2$; $x_2 = 7$ $y_1 = -4$; $y_2 = 0$	
21	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -7$; $x_2 = 2$ $y_1 = -2$; $y_2 = 2$	
22	Primitives: GL_POINTS, GL_LINES $x_1 = -3$; $x_2 = 6$ $y_1 = 0$; $y_2 = 4$	
23	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -4$; $x_2 = 5$ $y_1 = -1$; $y_2 = 3$	
24	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8$; $x_2 = 1$ $y_1 = -3$; $y_2 = 1$	

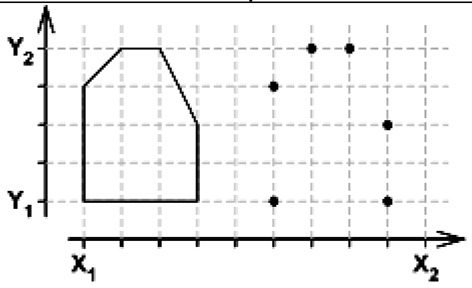
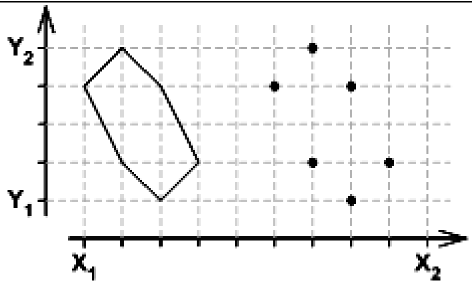
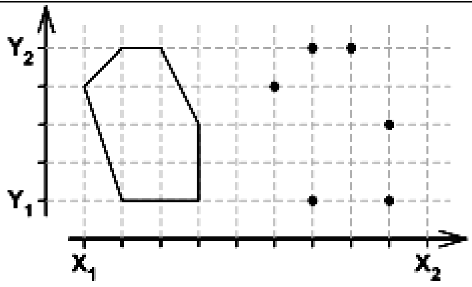
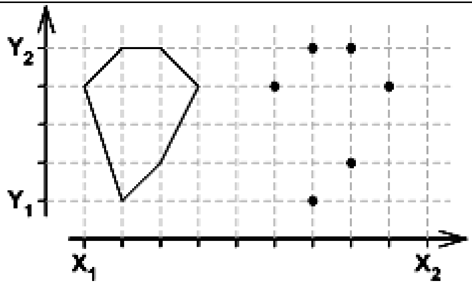
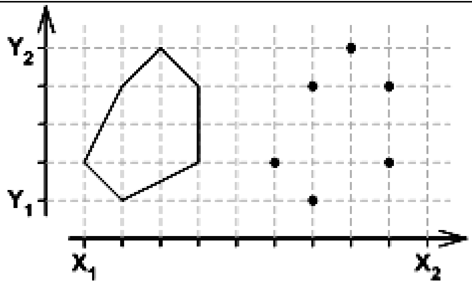
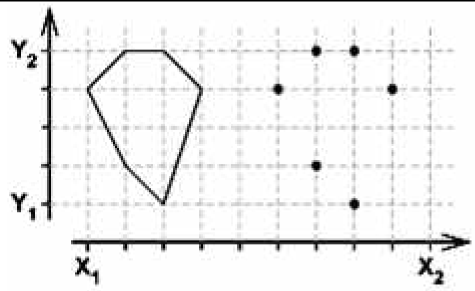
Continuation of Table 1.2

No.	Parameters	Shape
25	Primitives: GL_POINTS, GL_LINES $x_1 = -1.5$; $x_2 = 7.5$ $y_1 = -0.5$; $y_2 = 3.5$	
26	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2.5$; $x_2 = 6.5$ $y_1 = -2.5$; $y_2 = 1.5$	
27	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -9.5$; $x_2 = -0.5$ $y_1 = -3.5$; $y_2 = 0.5$	
28	Primitives: GL_POINTS, GL_LINES $x_1 = -5.5$; $x_2 = 3.5$ $y_1 = -1.5$; $y_2 = 2.5$	
29	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -7.5$; $x_2 = 1.5$ $y_1 = -3.5$; $y_2 = 0.5$	
30	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8.5$; $x_2 = 0.5$ $y_1 = -1.5$; $y_2 = 2.5$	

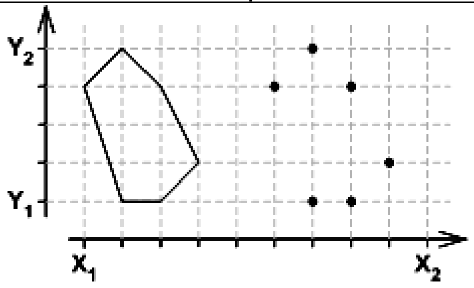
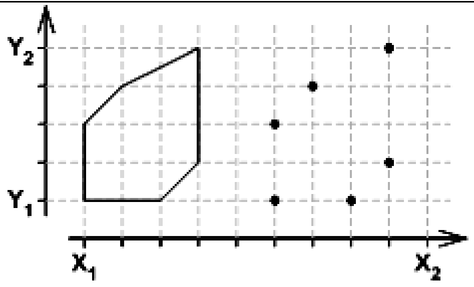
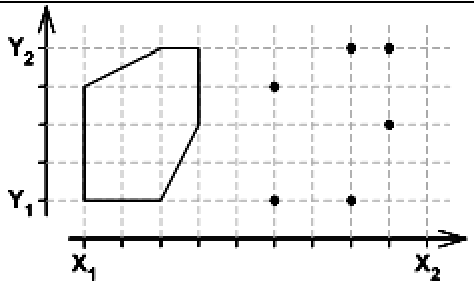
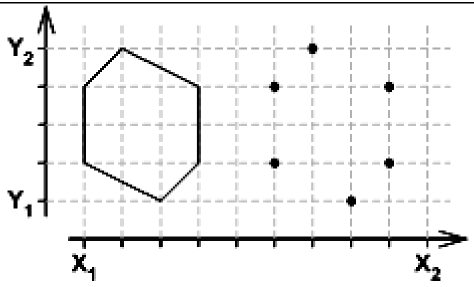
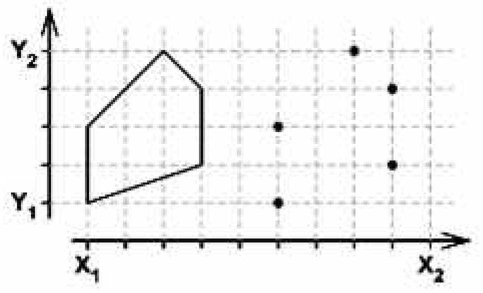
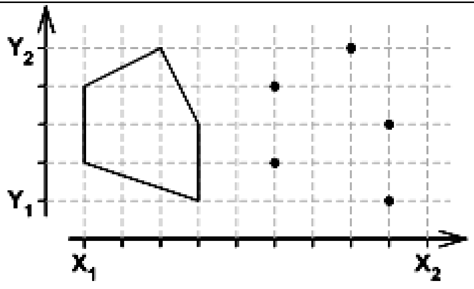
Continuation of Table 1.2

No.	Parameters	Shape
31	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -7.5$; $x_2 = 1.5$ $y_1 = -3.5$; $y_2 = 0.5$	
32	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8.5$; $x_2 = 0.5$ $y_1 = -1.5$; $y_2 = 2.5$	
33	Primitives: GL_POINTS, GL_LINES $x_1 = 0$; $x_2 = 9$ $y_1 = 0$; $y_2 = 4$.	
34	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -1$; $x_2 = 8$ $y_1 = -2$; $y_2 = 2$	
35	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8$; $x_2 = 1$ $y_1 = -3$; $y_2 = 1$	
36	Primitives: GL_POINTS, GL_LINES $x_1 = -4$; $x_2 = 5$ $y_1 = -1$; $y_2 = 3$	

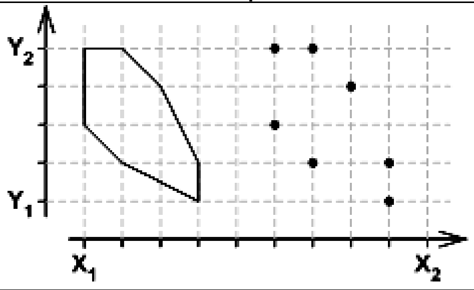
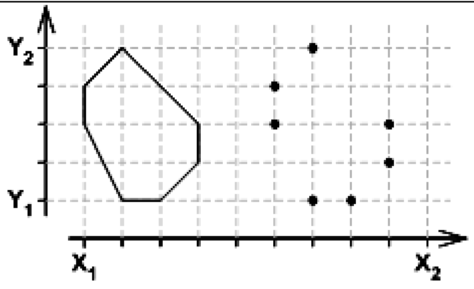
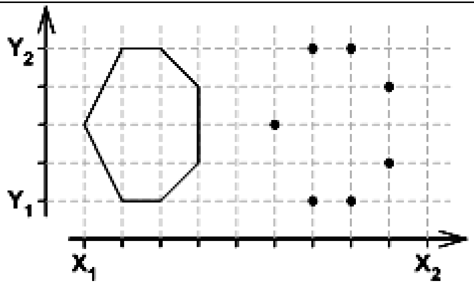
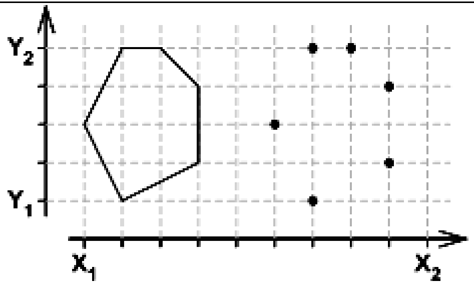
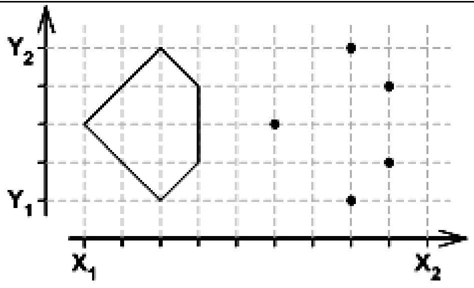
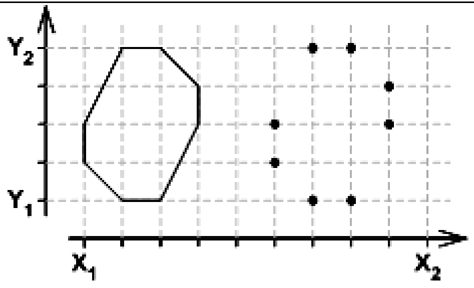
Continuation of Table 1.2

No.	Parameters	Shape
37	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -9$; $x_2 = 0$ $y_1 = -4$; $y_2 = 0$	
38	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -7$; $x_2 = 2$ $y_1 = -1$; $y_2 = 3$	
39	Primitives: GL_POINTS, GL_LINES $x_1 = 0$; $x_2 = 18$ $y_1 = 0$; $y_2 = 8$	
40	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2$; $x_2 = 16$ $y_1 = -4$; $y_2 = 4$	
41	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -16$; $x_2 = 2$ $y_1 = -6$; $y_2 = 2$	
42	Primitives: GL_POINTS, GL_LINES $x_1 = -8$; $x_2 = 10$ $y_1 = -2$; $y_2 = 6$	

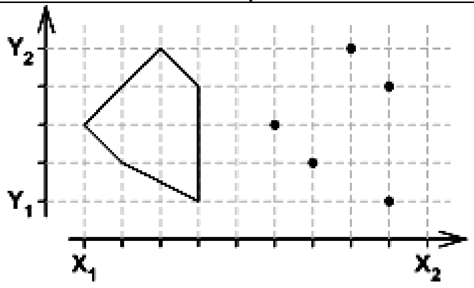
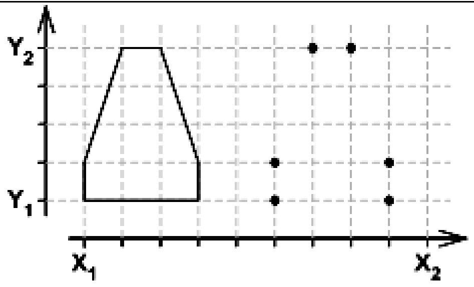
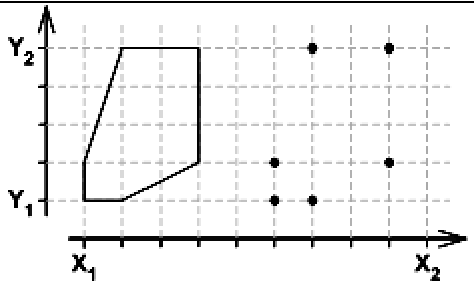
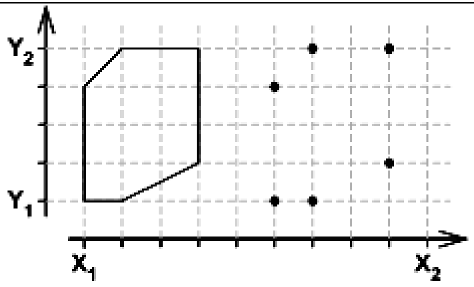
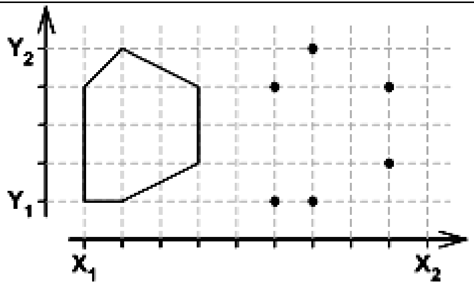
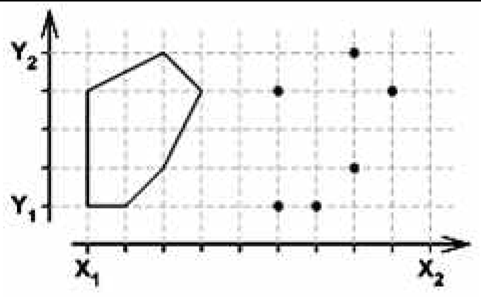
Continuation of Table 1.2

No.	Parameters	Shape
43	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -18$; $x_2 = 0$ $y_1 = -8$; $y_2 = 0$	
44	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -14$; $x_2 = 4$ $y_1 = -2$; $y_2 = 6$	
45	Primitives: GL_POINTS, GL_LINES: $x_1 = 0$; $x_2 = 4.5$ $y_1 = 0$; $y_2 = 2$	
46	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -0.5$; $x_2 = -4$ $y_1 = -1$; $y_2 = 1$	
47	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -4$; $x_2 = 0.5$ $y_1 = -1.5$; $y_2 = 0.5$	
48	Primitives: GL_POINTS, GL_LINES $x_1 = -2$; $x_2 = 2.5$ $y_1 = -0.5$; $y_2 = 1.5$	

Continuation of Table 1.2

No.	Parameters	Shape
49	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -4.5$; $x_2 = 0$ $y_1 = -2$; $y_2 = 0$	
50	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -3.5$; $x_2 = 1$ $y_1 = -0.5$; $y_2 = 1.5$	
51	Primitives: GL_POINTS, GL_LINES $x_1 = -1$; $x_2 = 8$ $y_1 = -2$; $y_2 = 2$	
52	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2$; $x_2 = 7$ $y_1 = -4$; $y_2 = 0$	
53	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -7$; $x_2 = 2$ $y_1 = -2$; $y_2 = 2$	
54	Primitives: GL_POINTS, GL_LINES $x_1 = -3$; $x_2 = 6$ $y_1 = 0$; $y_2 = 4$	

End of Table 1.2

No.	Parameters	Shape
55	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -4$; $x_2 = 5$ $y_1 = -1$; $y_2 = 3$	
56	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -8$; $x_2 = 1$ $y_1 = -3$; $y_2 = 1$	
57	Primitives: GL_POINTS, GL_LINES $x_1 = -1.5$; $x_2 = 7.5$ $y_1 = -0.5$; $y_2 = 3.5$	
58	Primitives: GL_POINTS, GL_LINE_STRIP $x_1 = -2.5$; $x_2 = 6.5$ $y_1 = -2.5$; $y_2 = 1.5$	
59	Primitives: GL_POINTS, GL_LINE_LOOP $x_1 = -9.5$; $x_2 = -0.5$ $y_1 = -3.5$; $y_2 = 0.5$	
60	Primitives: GL_POINTS, GL_LINES $x_1 = -5.5$; $x_2 = 3.5$ $y_1 = -1.5$; $y_2 = 2.5$	

Practical work № 2

OPENGL GRAPHICS PRIMITIVES

Aim of work: explore the concept of tessellation and learn how to use OpenGL graphics primitives to create surfaces. Master the handling of keyboard and mouse events to create interactive applications

Task

Using the tools specified by the instructor and taking into account the requirements given in Table 2.1, create software project with under OpenGL support. Use the **glOrtho** / **gluOrtho2D** and **glViewport** commands to set the isotropic coordinate system for the stagenat, taking into account the size of the figure specified in the variant (Table 2.2). After starting the application one tile in the workspace should be displayed. An example of the initial state of the application is shown in Fig. 2.1.

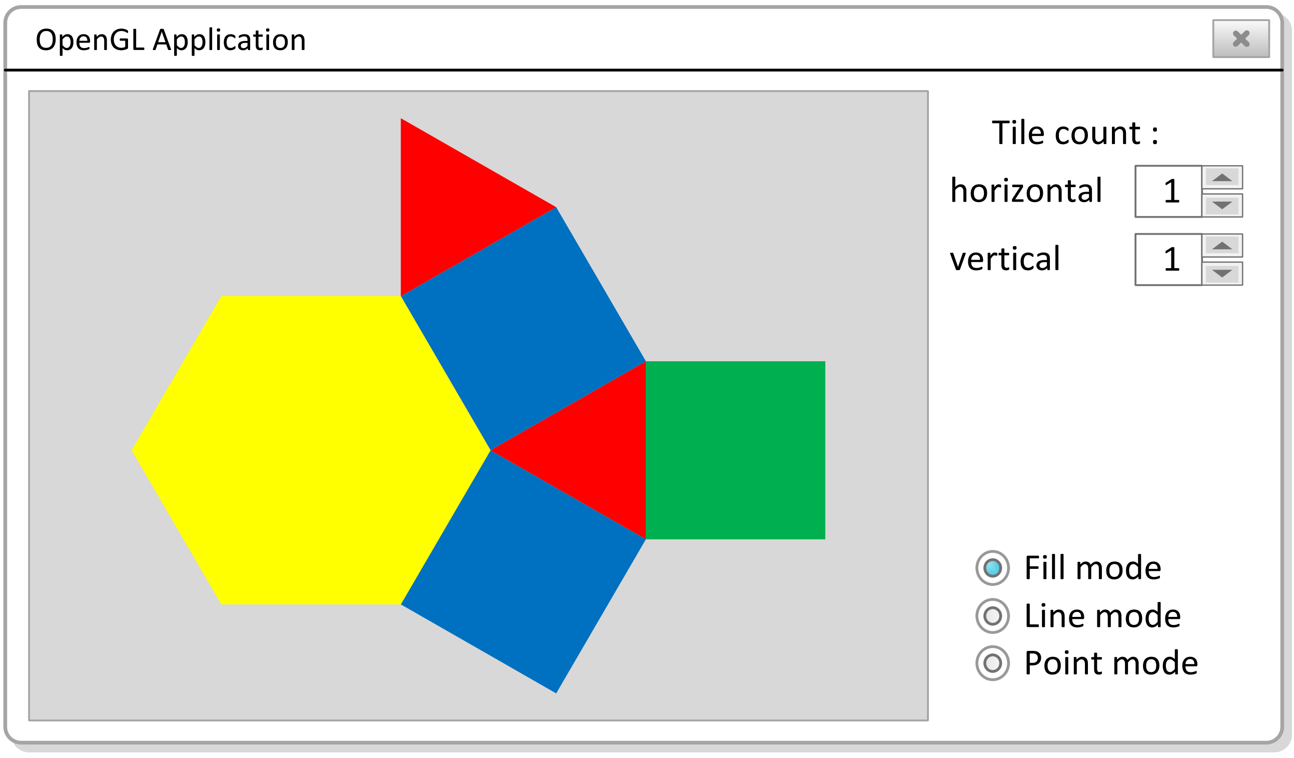
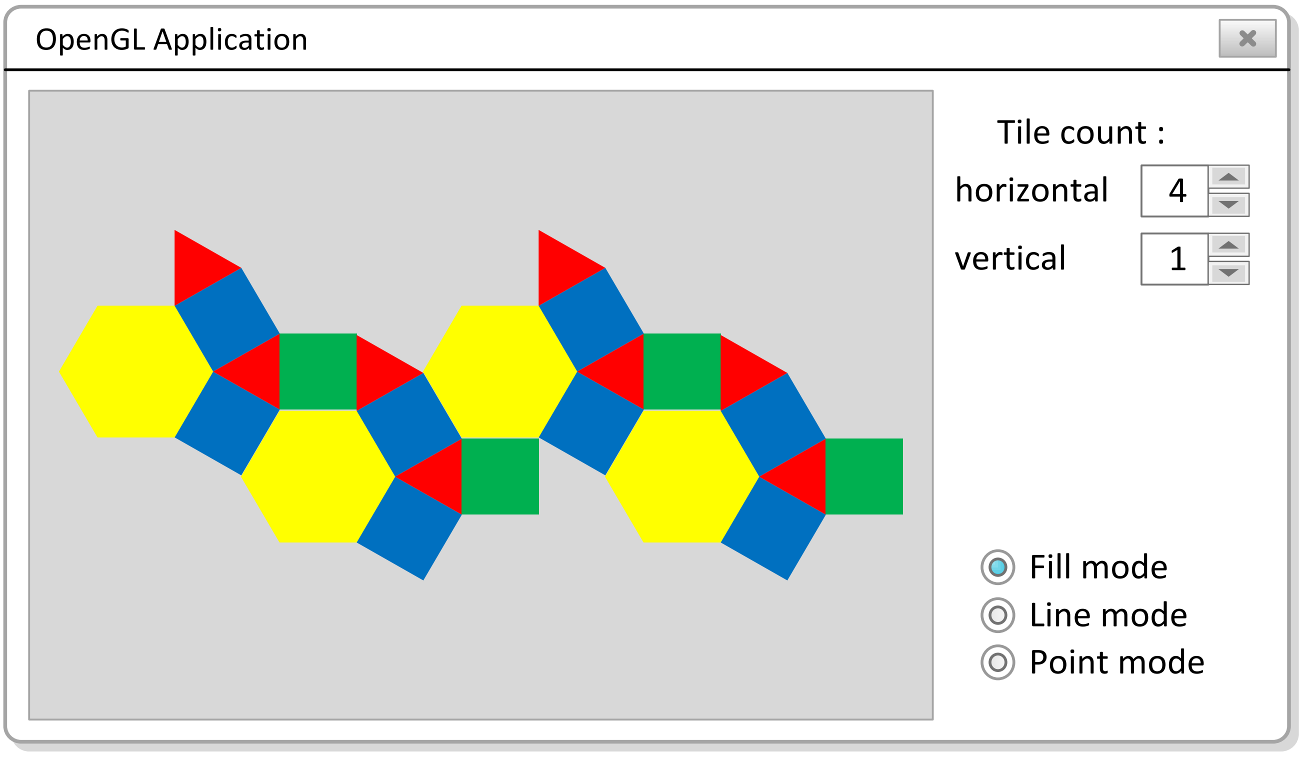


Fig. 2.1. Application view after start

All variants of tasks are based on regular polygons, the size of which is determined by the size of one edge. Six colors are supposed to be used for shading: white, gray (35 %), red, green, blue and yellow.

Using the keyboard or mouse, the user should be able to tessellate, tilling the work area horizontally and vertically [5]. In this case, the coordinate system must be adjusted so that the paved surface is located in the center of the work area. An example of application workspace tiling is shown in Fig. 2.2.


OpenGL Application

Tile count :

horizontal 4

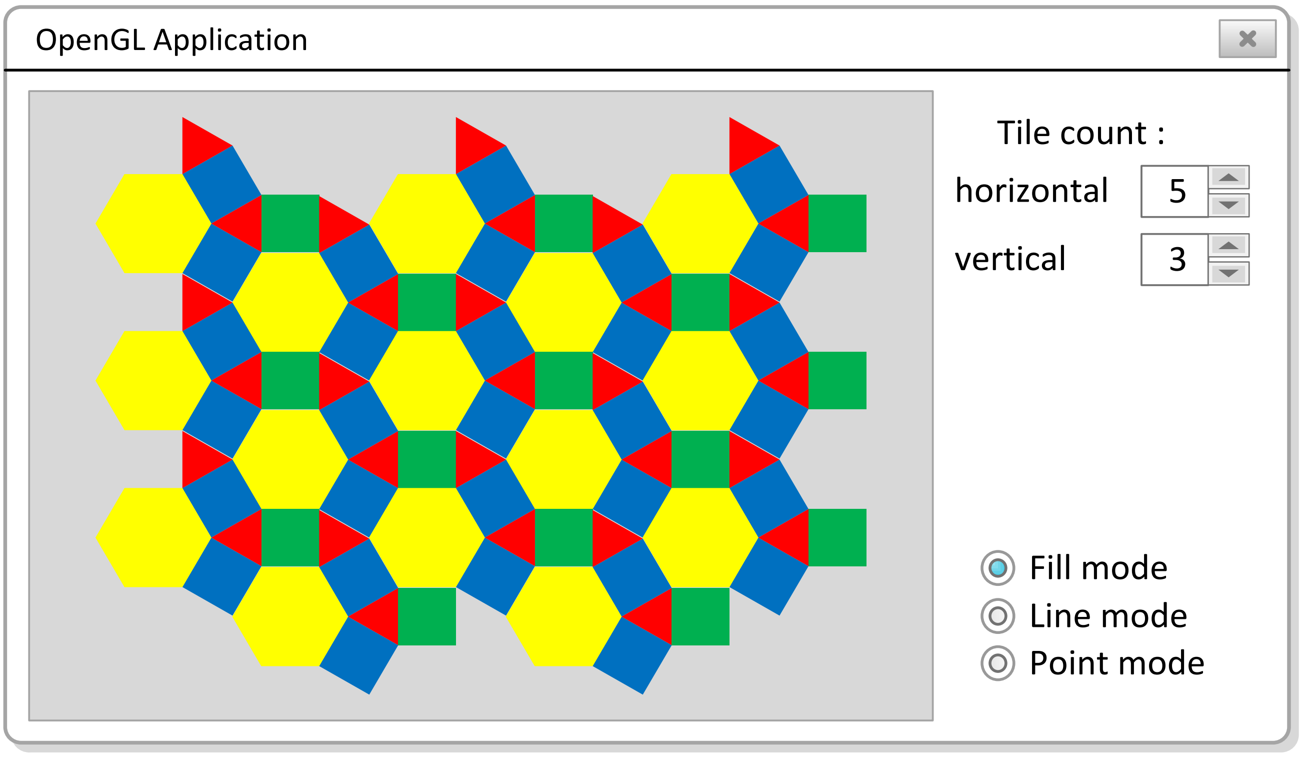
vertical 1

[✓ Fill mode]

Line mode

Point mode

a


OpenGL Application

Tile count :

horizontal 5

vertical 3

[✓ Fill mode]

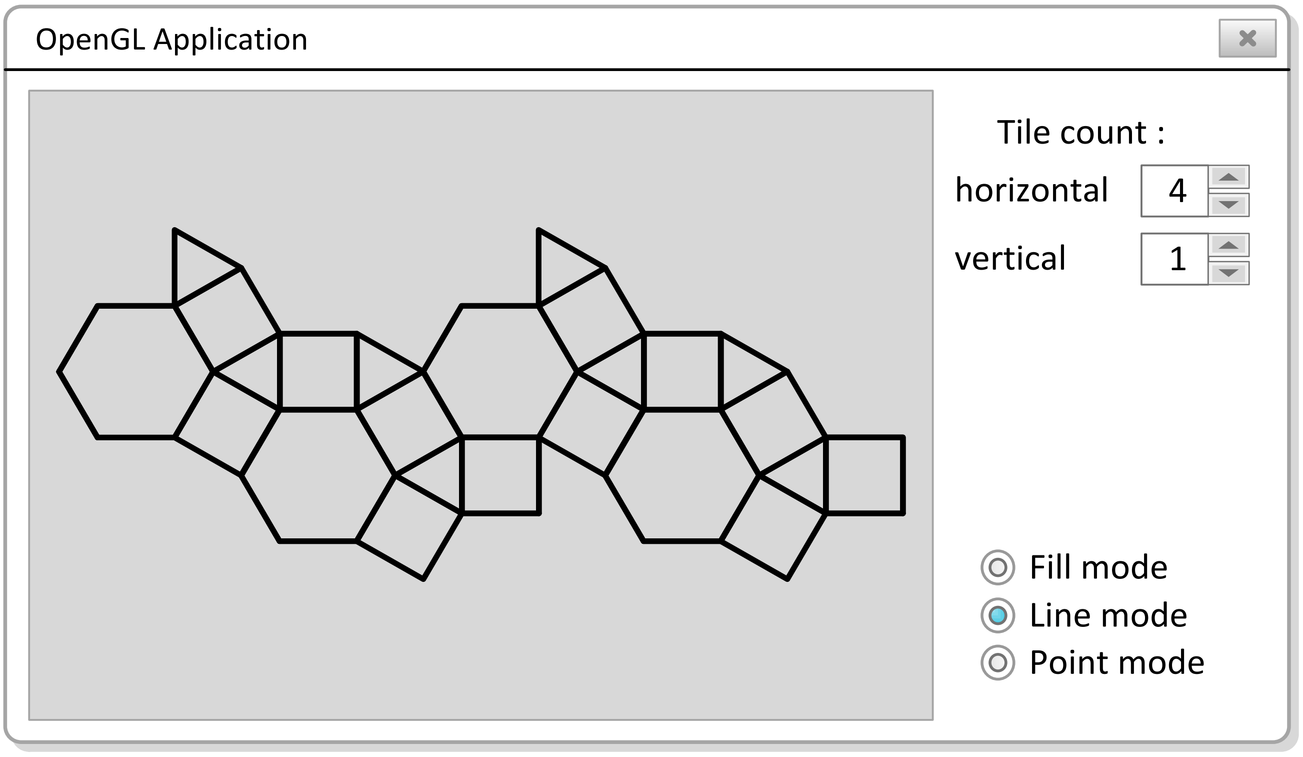
Line mode

Point mode

b

Fig. 2.2. Application view when tiled:
a – only horizontally; b – horizontally and vertically

In addition, the user should be able to change the display mode of OpenGL graphic primitives: point (only the vertices of the shape), outline (Fig. 2.3) and filled with color (see Fig. 2.2). It is assumed that switching between modes is performed by an event from the keyboard and / or the mouse. In this case, you can use both standard controls and your own, which are implemented and displayed using OpenGL (for an increased level of complexity, see Table 2.1).



Tile count :

horizontal 4

vertical 1

Fill mode

[✓ Line mode]

Point mode

Fig. 2.3. An example of controlling the output mode of graphic primitives

Methodical instructions

When displaying images, you should keep in mind that each surface of the OpenGL graphics primitive has two sides and the output mode for each of them can be configured separately using the glPolygonMode command.

To change the modes (model) of painting use the glShadeModel command. If the grayscale shading mode is disabled, the primitive color is determined by the color of only one vertex. For example, for GL_TRIANGLE_STRIP, the color of the first triangle is determined by the color of the third vertex, the second – by the fourth vertex, and so on.

To set the fill pattern, you must use the glEnable / glDisable toggle commands (as for the line pattern).

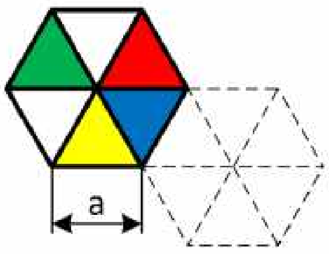
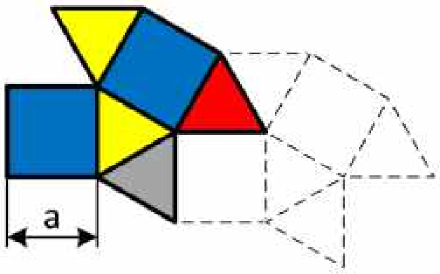
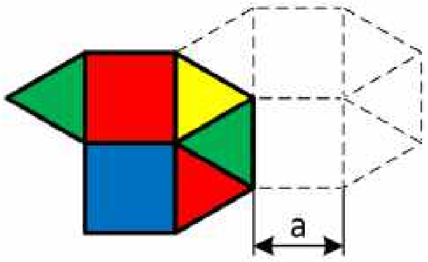
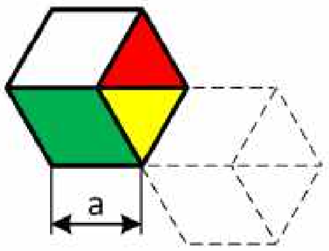
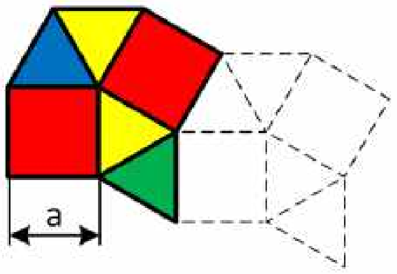
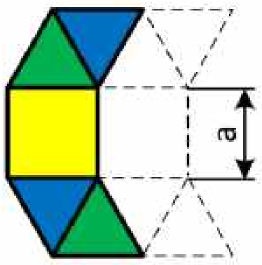
Checklist Questions

1. How does primitive coloring depend on vertex color and coloring mode?
2. How are the vertex traversal order and the primitive output mode related?
3. What is the difference and how are the inner and outer edges indicated?
4. How to programmatically implement isotropic and anisotropic coordinate systems?
5. How can I determine the current output mode of primitives?
6. How to set the fill pattern of a primitive?
7. How to find out if a fill pattern (lines) is being used or not?

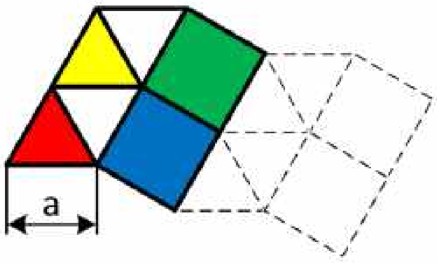
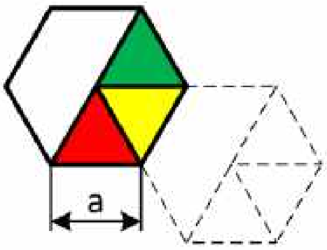
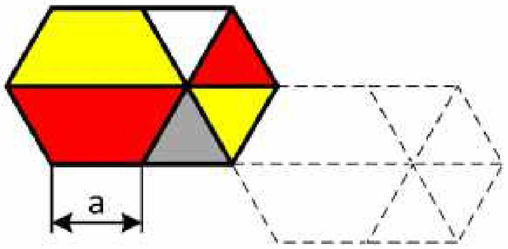
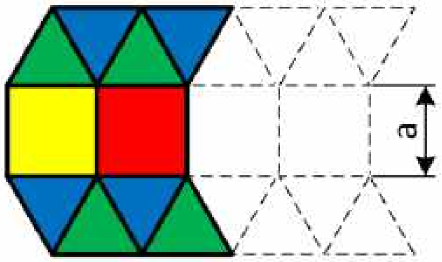
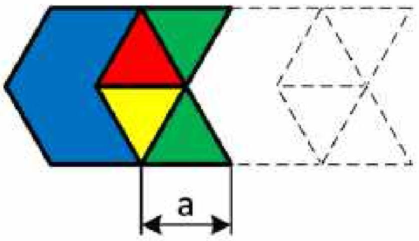
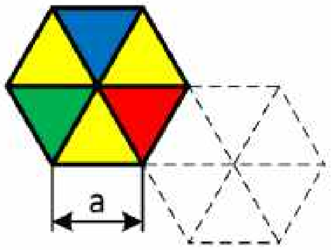
Table 2.1

No.	Complexity	Assignments	Points
1	Basic	When the application starts, the image corresponds to the task variant	1
2		Correct display of the task when changing both the size / position of the window and the tiling parameters	2
3		Organization of interaction with the user using one of the standard tools (keyboard, mouse, etc.)	1
4		Application of the minimum (within the variant) number of graphic primitives to complete the task	1
5	Advanced	Creating your own UI elements with OpenGL	2
6		Using OOP (developing your own classes)	1

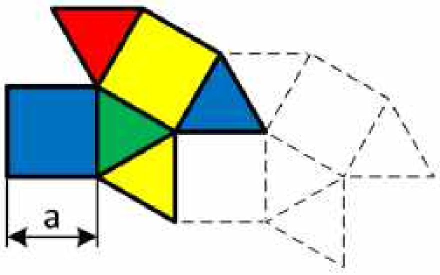
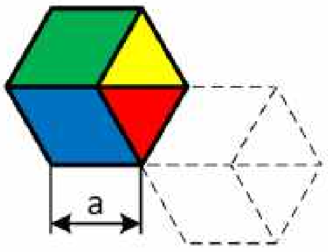
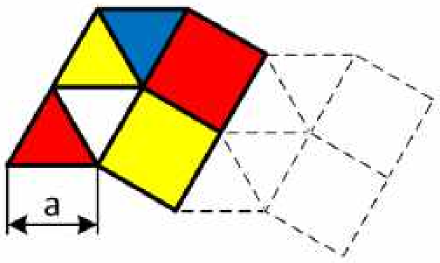
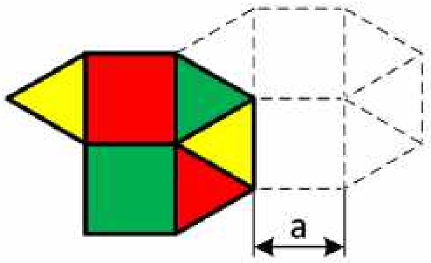
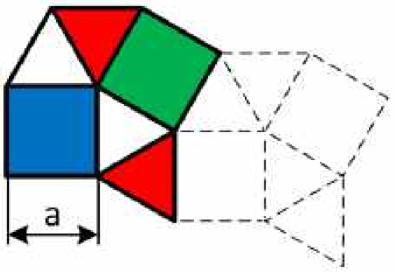
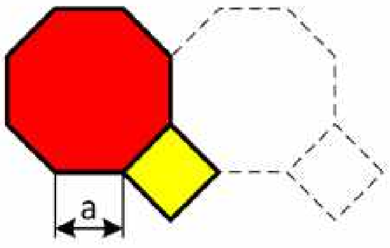
Table 2.2

No.	Parameters	Shape
1	Side ***a*** = 7 Primitive (s): GL_TRIANGLE_STRIP	
2	Side a = 4.25 Primitive (s): GL_TRIANGLES, GL_QUADS	
3	Side a = 75 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
4	Side a = 125 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
5	Side a = 12 Primitive (s): GL_TRIANGLES, GL_QUADS	
6	Side a = 0.05 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON,	

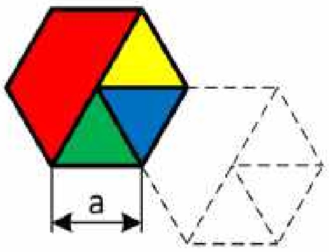
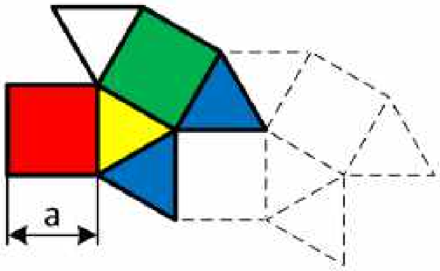
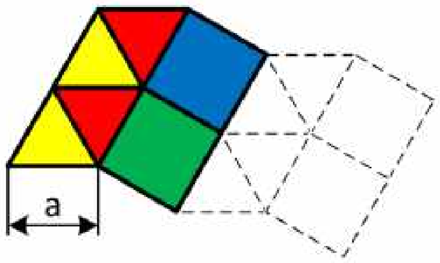
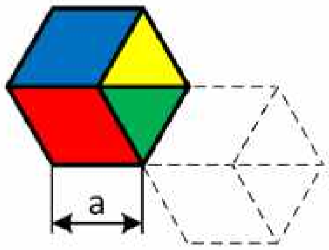
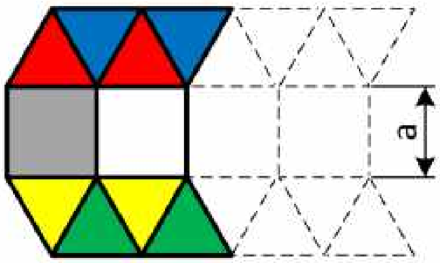
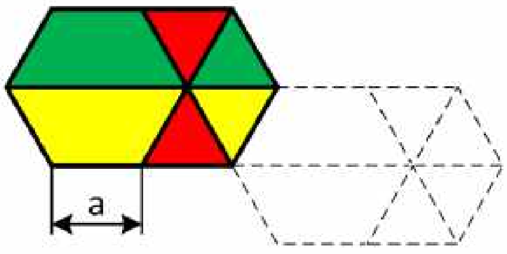
Continuation of Table 2.2

No.	Parameters	Shape
7	Side ***a*** = 50 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
8	Side a = 150 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
9	Side a = 8.5 Primitive (s): GL_TRIANGLES, GL_QUAD_STRIP	
10	Side a = 10 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
11	Side a = 75 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
12	Side ***a*** = 100 Primitive (s): GL_TRIANGLES	

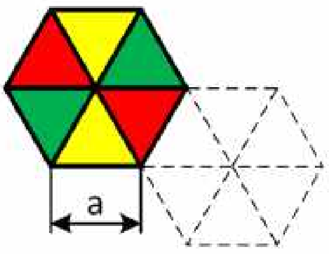
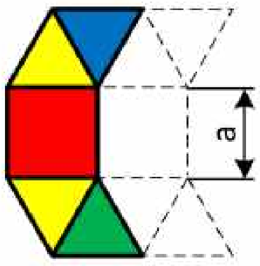
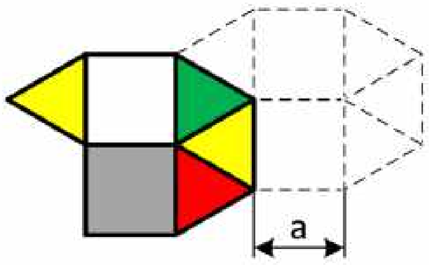
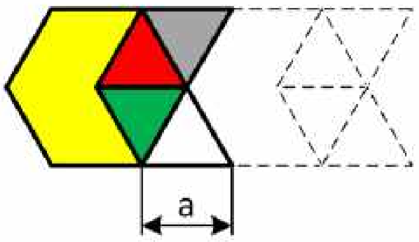
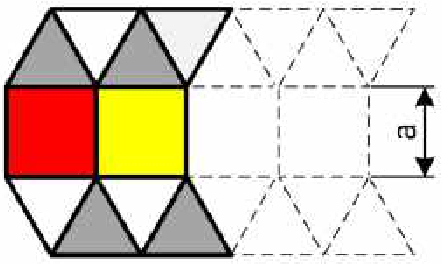
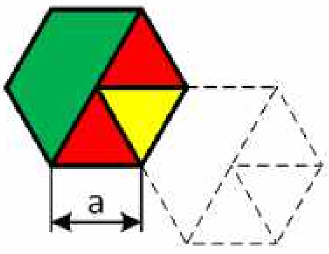
Continuation of Table 2.2

No.	Parameters	Shape
13	Side a = 0.2 Primitive (s): GL_POLYGON	
14	Side a = 1000 Primitive (s): GL_TRIANGLES, GL_QUAD_STRIP	
15	Side ***a*** = 30 Primitive (s): GL_TRIANGLES, GL_QUAD_STRIP	
16	Side a = 5 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
17	Side a = 8.5 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
18	Side a = 100 Primitive (s): GL_POLYGON, GL_QUADS	

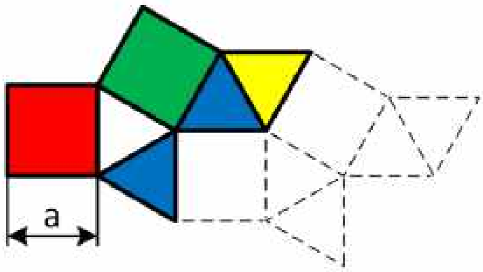
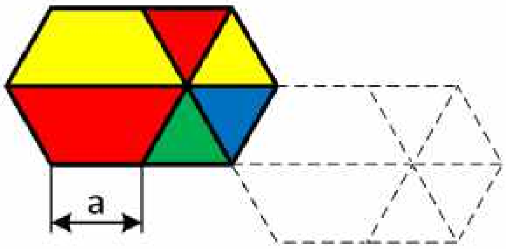
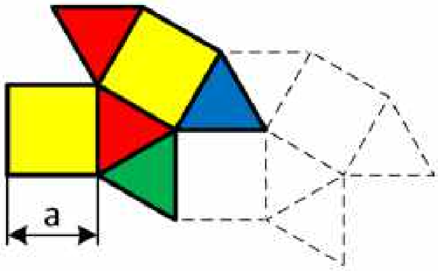
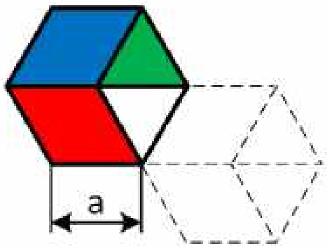
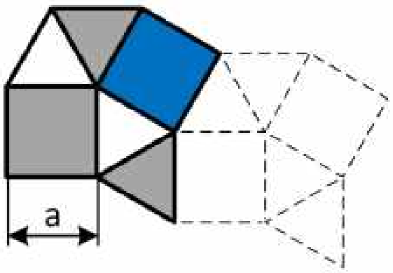
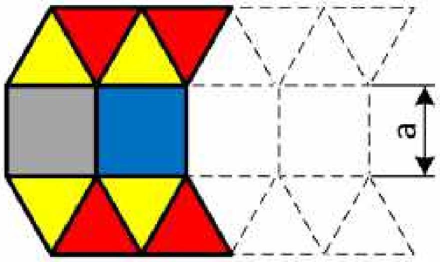
Continuation of Table 2.2

No.	Parameters	Shape
19	Side a = 0.15 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
20	Side a = 1500 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	
21	Side ***a*** = 25 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	
22	Side a = 250 Primitive (s): GL_POLYGON	
23	Side a = 20 Primitive (s): GL_TRIANGLES, GL_QUAD_STRIP	
24	Side a = 10 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	

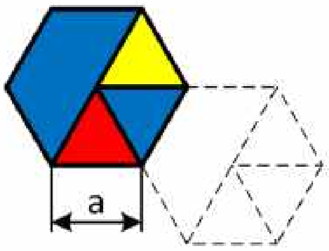
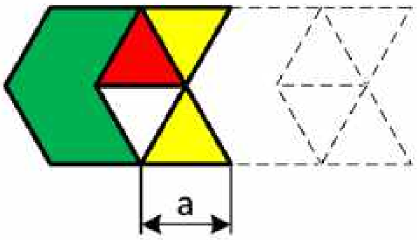
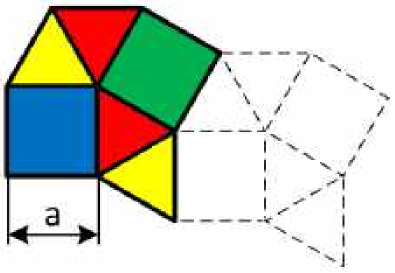
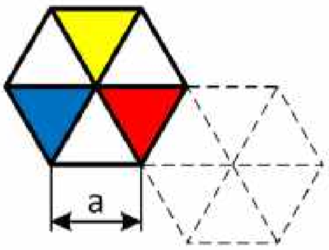
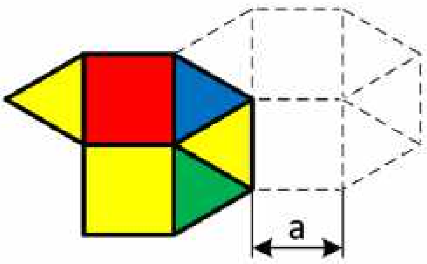
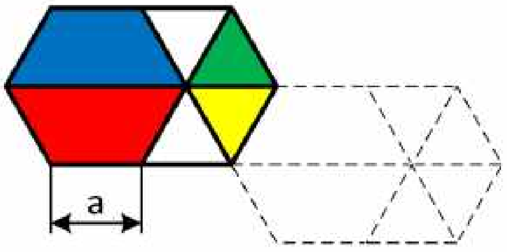
Continuation of Table 2.2

No.	Parameters	Shape
25	Side ***a*** = 3 Primitive (s): GL_TRIANGLE_FAN	
26	Side a = 20 Primitive (s): GL_POLYGON	
27	Side a = 30 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
28	Side a = 0.75 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
29	Side a = 0.25 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
30	Side a = 0.5 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	

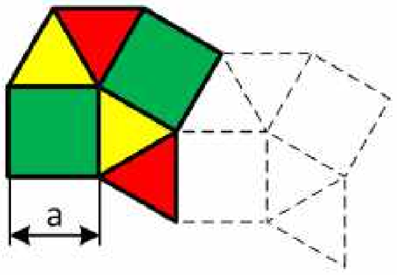
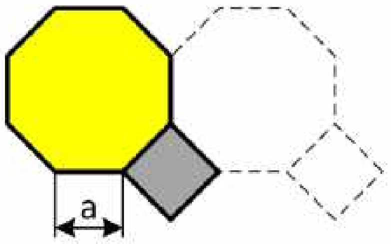
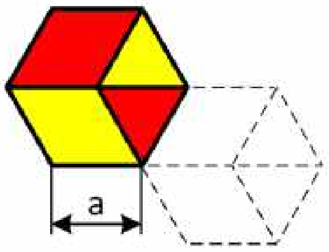
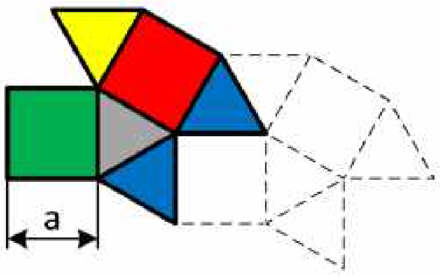
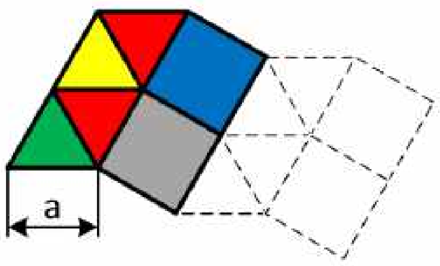
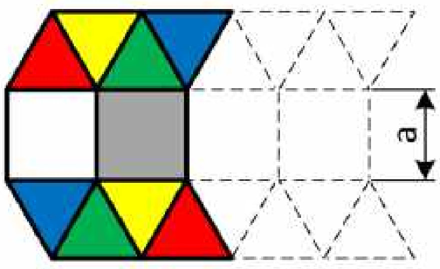
Continuation of Table 2.2

No.	Parameters	Shape
31	Side a = 1 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	
32	Side a = 375 Primitive (s): GL_TRIANGLE_FAN, GL_QUAD_STRIP	
33	Side a = 50 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
34	Side a = 40 Primitive (s): GL_TRIANGLE_STRIP, GL_QUAD_STRIP	
35	Side a = 1500 Primitive (s): GL_POLYGON	
36	Side a = 2.5 Primitive (s): GL_TRIANGLE_FAN, GL_QUAD_STRIP	

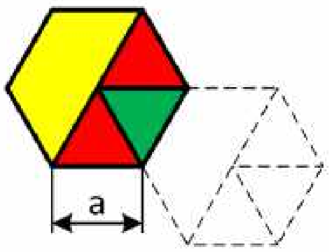
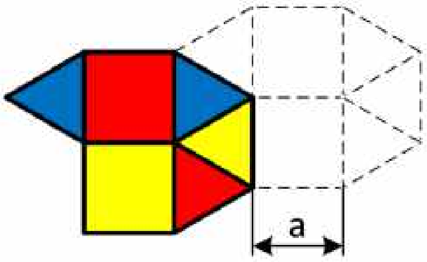
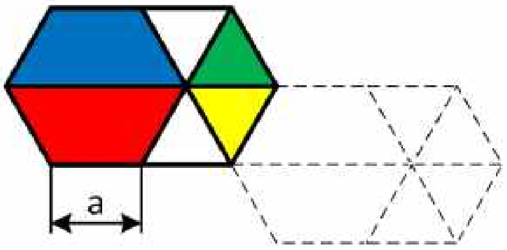
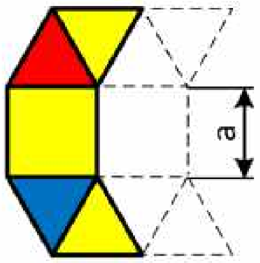
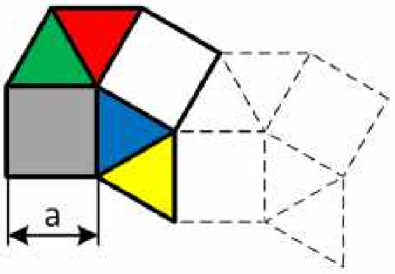
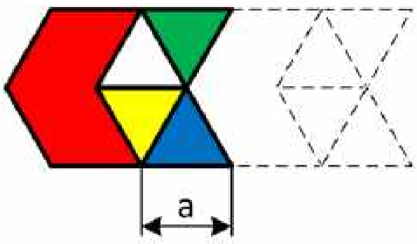
Continuation of Table 2.2

No.	Parameters	Shape
37	Side a = 2.5 Primitive (s): GL_TRIANGLES, GL_POLYGON	
38	Side a = 15 Primitive (s): GL_TRIANGLES, GL_POLYGON	
39	Side a = 0.75 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
40	Side ***a*** = 5 Primitive (s): GL_POLYGON	
41	Side a = 0.01 Primitive (s): GL_TRIANGLE_STRIP, GL_QUAD_STRIP	
42	Side a = 75 Primitive (s): GL_TRIANGLE_STRIP, GL_QUAD_STRIP	

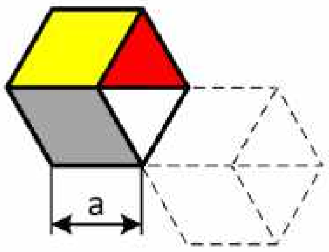
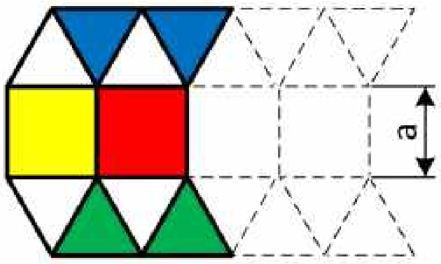
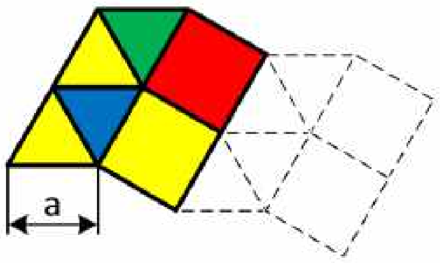
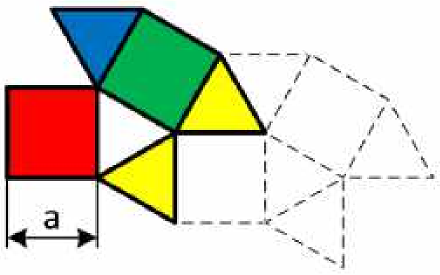
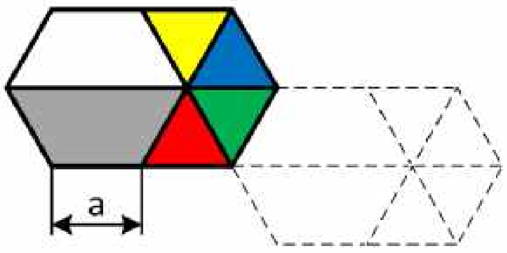
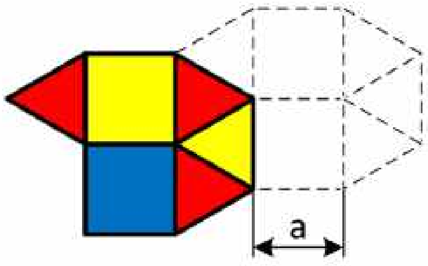
Continuation of Table 2.2

No.	Parameters	Shape
43	Side a = 2.25 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	
44	Side a = 5.5 Primitive (s): GL_POLYGON	
45	Side a = 0.01 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
46	Side a = 0.375 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
47	Side a = 12 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
48	Side a = 0.75 Primitive (s): GL_POLYGON	

Continuation of Table 2.2

No.	Parameters	Shape
49	Side a = 45 Primitive (s): GL_POLYGON	
50	Side a = 400 Primitive (s): GL_TRIANGLE_FAN, GL_QUAD_STRIP	
51	Side a = 2.25 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
52	Side a = 600 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
53	Side a = 3.5 Primitive (s): GL_TRIANGLE_FAN, GL_POLYGON	
54	Side a = 0.05 Primitive (s): GL_POLYGON	

End of Table 2.2

No.	Parameters	Shape
55	Side a = 300 Primitive (s): GL_TRIANGLE_FAN, GL_QUAD_STRIP	
56	Side a = 15 Primitive (s): GL_TRIANGLE_STRIP, GL_QUAD_STRIP	
57	Side ***a*** = 0.1 Primitive (s): GL_TRIANGLE_FAN, GL_QUADS	
58	Side a = 3.5 Primitive (s): GL_TRIANGLE_STRIP, GL_POLYGON	
59	Side a = 3.75 Primitive (s): GL_POLYGON, GL_QUAD_STRIP	
60	Side a = 1.25 Primitive (s): GL_TRIANGLE_STRIP, GL_QUADS	

Practical work № 3

SINGLE VARIABLE FUNCTION GRAPH

Aim of work: explore the basic concepts and principles of coordinate transformation for building a two-dimensional graph.

Task

Using the teacher specified development tools, develop a program for plotting a function of the form $y = f(x)$ on an arbitrary interval from $\boldsymbol{X_{min}}$ to $\boldsymbol{X_{max}}$ and display the points of intersection of the function with the abscissa axis. In addition, the program must have the following capabilities (Table 3.1):

- allow the user to set an interval from $\boldsymbol{X_{min}}$ to $\boldsymbol{X_{max}}$ with checking $\boldsymbol{X_{min}} < \boldsymbol{X_{max}}$;
- perform automatic scaling along the Y-axis for a user-specified interval from $\boldsymbol{X_{min}}$ to $\boldsymbol{X_{max}}$ (in addition, a manual mode for setting $\boldsymbol{Y_{min}}$ and $\boldsymbol{Y_{max}}$ is allowed);
- display the coordinate axes (and / or the coordinate grid) with the output of the values $\boldsymbol{X_{min}}$, $\boldsymbol{X_{max}}$, $\boldsymbol{Y_{min}}$ and $\boldsymbol{Y_{max}}$, of the boundaries of the visible area $\boldsymbol{X_{min}}$, $\boldsymbol{X_{max}}$, $\boldsymbol{Y_{min}}$ and $\boldsymbol{Y_{max}}$, while the coordinate system must be anisotropic;
- display all points where $f(x) = 0$, if they are in the specified interval from $\boldsymbol{X_{min}}$ to $\boldsymbol{X_{max}}$.

An example user interface is shown in Fig. 3.1. Function variants $f_1(x)$ of the basic level of complexity are shown in the Table 3.2.

For an increased level of complexity, it is necessary to additionally implement the correct output of the function $f_2(x)$, taking into account the scope of the function definition (options are indicated in Table 3.3) and display the break lines (Fig. 3.2).

Methodical instructions

After the user has determined the interval along the X axis in the dialog mode, you should set the number of points N required to plot the function graph. This number can be set by the user explicitly or obtained programmatically, for example, correspond to the width (number of pixels) of the work area. Based on this information, the step of the function argument is calculated:

$$h = \frac{X_{max} - X_{min}}{N - 1}$$

and the coordinates of the function points are calculated within a user-specified interval:

$$x_i = X_{min} + i \cdot h; \quad y_i = f(x_i)\,; \quad i \in 0 \ldots N - 1.$$

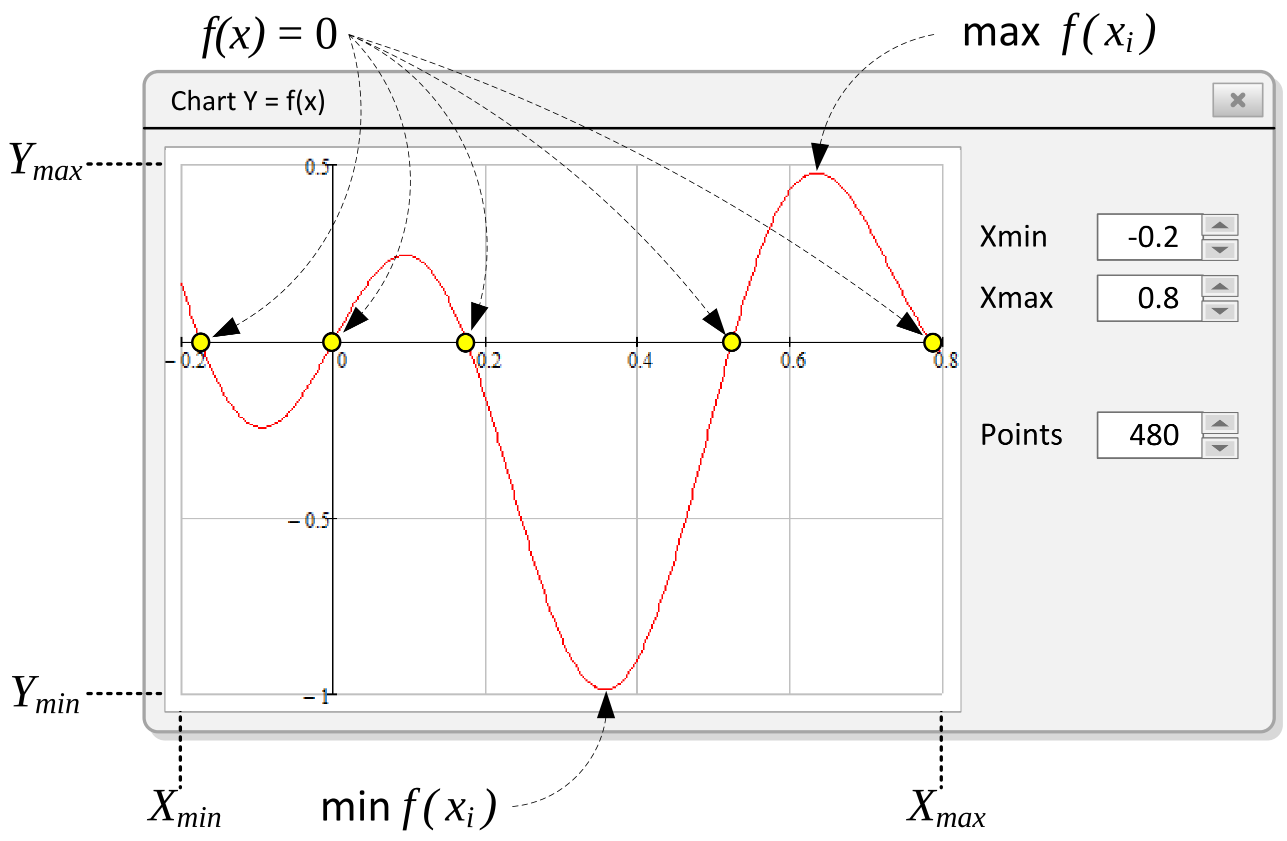


Fig. 3.1. An example of a basic functionality requirements for work

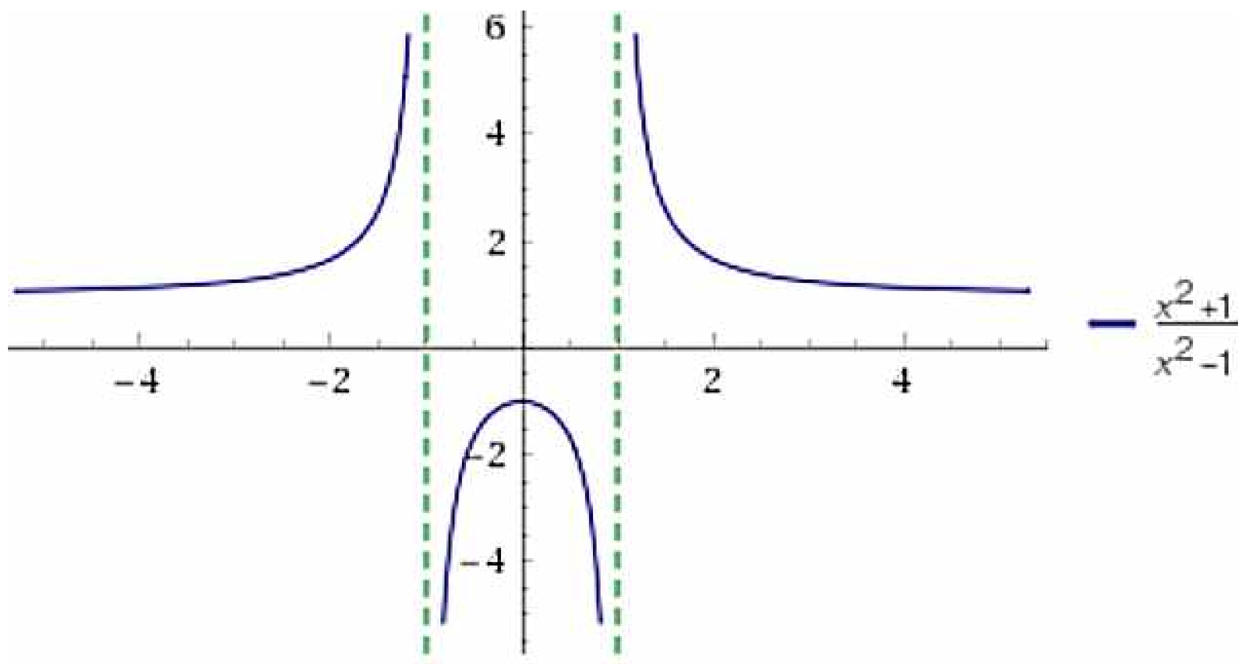


Fig. 3.2. Example function $f_2(x)$ and its break lines (shown by a dotted line)

Using these calculations, you can find the vertical boundaries of the work area:

$$Y_{min} = \min(y_i);$$
$$Y_{max} = \max(y_i);$$
$$i \in 0 \dots N-1.$$

Thus, knowing the values of the bounds of the interval along the ***X*** axis and calculating the values of the bounds along the ***Y*** axis, you can set the coordinate system (using the ***glOrtho(...)*** command) to display the graph $y = f(x)$ on the screen.

It is possible to find the roots x_0 of the function (points where $f(x_0) = 0$) based on the following property: if there is an intersection with the abscissa in the interval from x_i to x_{i+1}, then as a result of the product of the corresponding ordinates, the condition $f(x_i) \cdot f(x_{i+1}) \leq 0$ (Fig. 3.3) will be satisfied. In this case, the coordinates of the point of intersection of the function with the ***X*** axis are calculated simplified by the method of half division: $x_0 = (x_i + x_{i+1})/2$, $y_0 = f(x_0)$. If we take into account the discreteness of the screen and use the number of points to plot the function that is close to the width of the working area (in pixels) or exceeds it, then the simplified half-division method allows you to get a solution that is visually indistinguishable from the exact one.

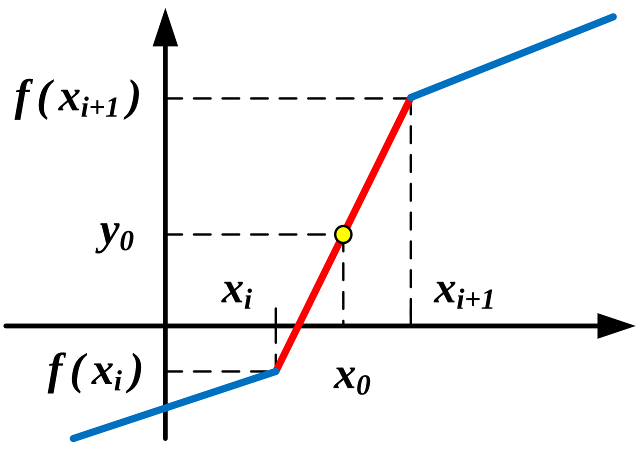


Fig. 3.3. Simplified version of the half division method

Students develop the algorithm for the correct display of a function $f_2(x)$, that has gaps in the definition area on the screen independently. Additional analysis of the features of your version of the function $f_2(x)$ can be obtained using the following electronic resources [7].

Checklist Questions

1. How to find out the number of pixels in the width and height of the screen?
2. How to determine the size of the client (work) area?
3. How is a forced redrawing of a window done?
4. What is the algorithm for finding the minimum / maximum of a function on an interval?
5. How to choose a step for plotting a function?
6. How is an isotropic coordinate system different from an anisotropic one?
7. How do I set up a coordinate system using OpenGL commands?
8. What units are used in the Windows coordinate system?

Table 3.1

No.	Complexity	Assignments	Points
1	Basic	The coordinate axes and the graph of the function f1 (x) are displayed on a user-specified interval from Xmin to Xmax and from Ymin to Ymax	1
2		Automatic calculation of Ymin and Ymax on a given interval from Xmin to Xmax of the function f1 (x)	2
3		Calculation and display of points f1 (x) = 0	2
4	Advanced	Correct display of the f2 (x) graph (without false display of breakpoints as points of intersection with the abscissa axis) and displaying the break lines of the function	2
5		Using OOP (inheritance, applyingvirtual and abstract methods)	1

Table 3.2

No.	Function ***f₁(x)***
1	$f_1(x) = 2\cos(0.1x + \cos x)$
2	$f_1(x) = \mathrm{tg}(\cos(2x + 0.1))$
3	$f_1(x) = \mathrm{arctg}(\cos^{13}(x + 2))$
4	$f_1(x) = \dfrac{\sin(x)}{\cos(2x) + 1.5}$
5	$f_1(x) = \mathrm{tg}(\cos(2x)) + \dfrac{\mathrm{tg}(\cos 5x)}{2}$

Continuation of Table 3.2

No.	Function ***f₁(x)***
6	$f_1(x) = (\sin(3x) + 1.5)^{\cos 2x} - 1$
7	$f_1(x) = \dfrac{\sin(x+1)}{\cos^2(4x) + \cos^3(3x) + 2}$
8	$f_1(x) = \sin^2(2x)\cos^3(3x)$
9	$f_1(x) = \dfrac{\cos 3x}{\cos(5x) + 1.1}$
10	$f_1(x) = 5\sin(0.2x + \sin x)$
11	$f_1(x) = \mathrm{tg}(1.25 \sin x)$
12	$f_1(x) = \dfrac{\cos(\pi x)}{(\sin(5\pi x/3) + 1.5)^3}$
13	$f_1(x) = \dfrac{\lvert\cos(0.5x + 1)\rvert \cos x}{\lvert\cos(x + 0.01)\rvert}$
14	$f_1(x) = \dfrac{\cos(x)}{\sqrt{\cos(6x) + 1.01}}$
15	$f_1(x) = \mathrm{ctg}(\sin(0.25x) + 1.05) - 2$
16	$f_1(x) = \cos(2x + 1) - 0.5 \cdot \sin(5x)$
17	$f_1(x) = \dfrac{\cos(3x + 1)}{(\cos(5x) + 1.21)^2}$
18	$f_1(x) = \sin^2(x + 1)\cos^3(2x - 1)$
19	$f_1(x) = \mathrm{tg}\dfrac{0.5\sin(2x)}{1.5 + \cos(5x)}$
20	$f_1(x) = 3\sin(0.2x + \sin 2x)$
21	$f_1(x) = \mathrm{tg}(1.3\sin(\cos(x) + x))$

Continuation of Table 3.2

No.	Function ***f₁(x)***
22	$f_1(x) = \sin(3\cos(x^2) + x)$
23	$f_1(x) = \dfrac{\cos(\pi x + \pi/4)}{\sqrt{\sin(7\pi x/5) + 1.01}}$
24	$f_1(x) = \dfrac{\sin(\pi x/2)}{\cos(\pi x) - \pi}$
25	$f_1(x) = \text{ctg}\left(1.25\sin(2x + \cos(4x)) + \frac{\pi}{2}\right)$
26	$f_1(x) = \text{tg}\dfrac{0.05 + sin\,x}{1.25 + cos\,x}$
27	$f_1(x) = \dfrac{\cos(x)}{\sqrt{\sin(3x) + 1.01}}$
28	$f_1(x) = \dfrac{1 - e^{2\sin x}}{1 + e^{3\cos(\pi x + 1)}}$
29	$f_1(x) = \text{ctg}(\sin(5x) + 1.15)$
30	$f_1(x) = 5\cos(0.2x - \sin x)$
31	$f_1(x) = \text{tg}(1.25\cos(2x + \cos(6x)))$
32	$f_1(x) = \dfrac{\sin^3(x - \pi/2)}{\cos^3(4x) + \cos^2(3x) + 2}$
33	$f_1(x) = \cos(\sin(2x + 1) + x)$
34	$f_1(x) = \dfrac{\cos 3x}{\sqrt{\cos(5x + 1.21)}}$
35	$f_1(x) = \text{ctg}\left(\sin(2x) - \frac{\pi}{2}\right)$
36	$f_1(x) = \text{tg}(\sin(x + \cos(2x)))$
37	$f_1(x) = 4\cos^4(\pi x)\,\sin^3(2\pi x)$

Continuation of Table 3.2

No.	Function ***f₁(x)***
38	$f_1(x) = \dfrac{1-e^{2\sin x}}{1+e^{3\cos(\pi x+1)}}$
39	$f_1(x) = \dfrac{\cos(x)}{\sin(x)+\frac{\pi}{2}}$
40	$f_1(x) = \pi \sin(0.3x-\cos x)$
41	$f_1(x) = \mathrm{tg}(1.5\cos(\cos(3x)+x))$
42	$f_1(x) = \dfrac{2\sin 2x}{(\cos x+1.5)}+1$
43	$f_1(x) = \arcsin(\cos(\pi x))$
44	$f_1(x) = (\cos(0.5x)+1.5)^{-(\sin(2\mathrm{x})+1.2)}-\dfrac{\pi}{2}$
45	$f_1(x) = \mathrm{ctg}\left(1.25\cos(2x+\cos(5x))+\dfrac{\pi}{2}\right)$
46	$f_1(x) = \dfrac{e^{\cos\lvert 3x\rvert}}{e^{\sin x}}-\pi$
47	$f_1(x) = \sin(\cos(2x)+x)$
48	$f_1(x) = \dfrac{\sin(x+1)}{e^{\cos 4x}}$
49	$f_1(x) = \mathrm{tg}\left(1.3\sin\left(x+2\cos\left(\dfrac{x}{2}\right)\right)\right)$
50	$f_1(x) = \sin(0.75x-\sin x)$
51	$f_1(x) = \mathrm{tg}(1.25\cos(2^x+\cos(6x)))$
52	$f_1(x) = \dfrac{\sin\left(3x+\frac{\pi}{4}\right)}{\sqrt{\cos(5x)+1.1}}$
53	$f_1(x) = \arccos(\sin(\pi x))-\dfrac{\pi}{2}$

End of Table 3.2

No.	Function ***f₁(x)***
54	$f_1(x) = \sin^2(3x)\ \cos^3(2x+1)$
55	$f_1(x) = \operatorname{ctg}\left(1.25\ \sin(2x+\sin(4x))+\frac{\pi}{2}\right)$
56	$f_1(x) = \frac{\cos 3x}{e^{\cos(2x-1.5)}}$
57	$f_1(x) = e^{\sin 3x-\cos\lvert x\rvert}-1$
58	$f_1(x) = \cos\left(\frac{x}{2}\right)-\frac{\pi}{2}+e^{\sin 2x\cdot\cos 5x}$
59	$f_1(x) = \operatorname{ctg}\left(\cos(x)+\frac{\pi}{2}\right)$
60	$f_1(x) = \operatorname{tg}(1.5\cos(\sin(3x)+2^x))$

Table 3.3

No.	Function ***f₂(x)***
1	$f_2(x) = \ln(\lvert\sin x\rvert)+\sin 3x$
2	$f_2(x) = \frac{\cos(\pi x)\ e^{\cos x}}{\lvert\cos(\pi x)\rvert}-0.5$
3	$f_2(x) = \operatorname{tg}\left(\frac{\pi}{2}\cos(\pi x)\right)$
4	$f_2(x) = \frac{\sin x}{\cos \pi x}$
5	$f_2(x) = \frac{\sin(\pi x)}{\ln(\cos(\pi x)+1)}$
6	$f_2(x) = \frac{\sin(\pi x+1)}{\lvert\sin(2\pi x)\rvert}+2\cos(\pi x)$
7	$f_2(x) = \frac{1}{\cos 3x}+\frac{1}{\sin 2x}$
8	$f_2(x) = \ln(\lvert\sin(\pi x)\rvert)+\cos(3.5\cdot\pi x)$

Continuation of Table 3.3

No.	Function ***f₂(x)***
9	$f_2(x) = e^{2\cos\pi x}\ \mathrm{tg}\frac{\pi x}{3}$
10	$f_2(x) = e^{-\sin(3\pi x)}\tan\left(\frac{\pi}{2}\cos(\pi x)\right)$
11	$f_2(x) = \frac{\cos(\pi x)}{\lvert\cos(\pi x)\rvert} + e^{\cos\left(\frac{\pi x}{2}\right)}$
12	$f_2(x) = \mathrm{tg}(2\sin(x))$
13	$f_2(x) = \frac{\cos(\pi x/2)\cos\lvert\pi x\rvert}{\lvert\sin(\pi x)\rvert}$
14	$f_2(x) = \ln(1+\cos(x))\ \lg(2+\sin(5x))$
15	$f_2(x) = e^{-\mathrm{tg}\,\pi x}\ \cos(3\pi x)$
16	$f_2(x) = \mathrm{tg}\left(\frac{\pi x}{3}\right)\frac{\cos(\pi x)}{\lvert\cos(\pi x)\rvert}$
17	$f_2(x) = \ln(\sin x + 1) + \cos(4x+1)$
18	$f_2(x) = \mathrm{tg}\left(\frac{\pi}{2}\cos(\pi x)\right) + \sin(2\pi x)$
19	$f_2(x) = e^{-1/(10\cdot\cos(\pi x))}\ \sin(\pi x)$
20	$f_2(x) = \frac{\sin(\pi x/2)\sin\lvert 3\pi x\rvert}{\lvert\sin(\pi x)\rvert}$
21	$f_2(x) = \mathrm{ctg}(x)\cos(\pi x)$
22	$f_2(x) = \sin(3x) + \ln(\cos(x)+1)$
23	$f_2(x) = \mathrm{tg}(e^x)$
24	$f_2(x) = \frac{\cos(\pi x)}{\lvert\cos(\pi x)\rvert} + \ln(\cos(\pi\,x/2)+1) + 1$
25	$f_2(x) = \mathrm{tg}(x)\cos(\pi x)$
26	$f_2(x) = \frac{\lvert\cos(\pi x)\rvert}{\cos(\pi x)} + 2\cos\left(\frac{\pi x}{2}\right)$

Continuation of Table 3.3

No.	Function ***f₂(x)***
27	$f_2(x) = \ln(\sin(\pi x) + 1) + e^{\sin(3\pi x)}$
28	$f_2(x) = \lg(\cos(2\pi x) + 1) + e^{\sin 3\pi x}$
29	$f_2(x) = \dfrac{1}{\ln(\sin(\pi x) + 1)} - e^{2\cos(3\pi x)}$
30	$f_2(x) = \dfrac{\cos(\pi x)\ \ln(\cos(\pi\, x/2) + 1)}{\lvert\cos(\pi x)\rvert}$
31	$f_2(x) = \lg(\sin(2x) + 1) + e^{2\cos x} - \pi$
32	$f_2(x) = \dfrac{\lvert\sin(\pi x)\rvert}{\sin(\pi x)} + 2\cos(\pi x)$
33	$f_2(x) = e^{-\cos(3\pi x)}\ \mathrm{tg}\left(\dfrac{\pi}{2}\sin(\pi x)\right)$
34	$f_2(x) = \mathrm{tg}\left(\dfrac{\pi}{2} + \sin(3x)\right) + 2\cos x$
35	$f_2(x) = \dfrac{\sin(2\pi x)}{\lvert\sin(\pi x)\rvert} - 2\sin(\pi x)$
36	$f_2(x) = \ln(\sin(x) + 1)\cos(4x + 1)$
37	$f_2(x) = e^{-\cos(4\pi x)}\ \mathrm{tg}\left(\dfrac{\pi}{2}\sin(\pi x)\right) + \cos(\pi x)$
38	$f_2(x) = \dfrac{\cos(3\pi x)}{\ln(\sin(\pi x) + 1)}$
39	$f_2(x) = \dfrac{\cos(\pi x)\ \ln(\sin(x/2) + 1)}{\lvert\cos(\pi x)\rvert}$
40	$f_2(x) = e^{\sin(4\pi x)} + \ln(\cos(\pi x) + 1)$
41	$f_2(x) = \dfrac{\lg(\cos(2\pi x) + 1)}{x}$
42	$f_2(x) = \dfrac{\sin(\pi x/2)\sin\lvert\pi x\rvert}{\lvert\sin(\pi x)\rvert}$
43	$f_2(x) = \mathrm{arctg}(\tan(\pi x) - 5\cos(\pi x))$
44	$f_2(x) = e^{-\cos(4\pi x)}\ \mathrm{ctg}\left(\dfrac{\pi}{2}\sin(\pi x)\right) - \sin(2\pi x)$

End of Table 3.3

No.	Function ***f₂(x)***
45	$f_2(x) = \operatorname{ctg}\left(\frac{\pi}{2}\sin(\pi x)\right) + \cos(2\pi x)$
46	$f_2(x) = \frac{(x^2+1)\cos(2\pi x)}{(x^2-1)\sin(2\pi x)}$
47	$f_2(x) = \frac{\cos(\pi x)\ e^{\sin x}}{\lvert\cos(\pi x)\rvert} + \frac{\pi}{2}$
48	$f_2(x) = \frac{(9-x^2)\ \cos(x)}{(9+x^2)\sin(x+1)}$
49	$f_2(x) = e^{\cos(4\pi x)}\ln(\sin(\pi x)+1)$
50	$f_2(x) = \frac{\sin(\pi x/2+2)\lvert\cos(3\pi x)\rvert}{\sin\lvert\pi x\rvert} + 0.5$
51	$f_2(x) = \frac{\sin(\pi x)}{\lvert\sin(\pi x)\rvert} + \ln(\sin(\pi\, x/3)+1) + 2$
52	$f_2(x) = e^{\operatorname{tg}(\pi x/2)} - 3\cos(\pi x)$
53	$f_2(x) = \frac{\cos(2\pi x)}{\ln(\cos(3\pi x)+1)}$
54	$f_2(x) = e^{\sin(4\pi x)}\ln(\cos(\pi x)+1)$
55	$f_2(x) = e^{\operatorname{tg}(\pi x/2)} - 5\cos(5\pi x)$
56	$f_2(x) = \operatorname{ctg}\left(\frac{\pi x}{2}\right)\frac{\cos(\pi x)}{\lvert\cos(\pi x)\rvert}$
57	$f_2(x) = \frac{\cos(\pi x/2)\cos\lvert\pi x\rvert}{\lvert\cos(\pi x)\rvert}$
58	$f_2(x) = \ln(\cos(3x)+1)\ \sin x$
59	$f_2(x) = \operatorname{ctg}\left(\frac{\pi}{2}\sin(3\pi x)\right)\ \cos(\pi x)$
60	$f_2(x) = e^{\operatorname{ctg}(\pi x)} - 5\sin(4\pi x)$

Practical work № 4

A CONIC SECTION CURVES

Aim of work: explore mathematical methods and tools for the implementation of graphical primitives.

Task

Using the teacher specified development tools, develop a program for displaying conic section curves on the screen (to a Windows form) using lines. The grading system is shown in Table 4.1, and the options for the tasks are in Table 4.2. For curves marked with "++" in the variant, find and display the intersection points, if any, with an arbitrary segment, the coordinates of which are set by the user.

Methodical instructions

Each curve of the second order can be represented as a sequence of line segments. In this case, the intersection of a second-order curve and an arbitrary segment can be considered as a search for a common point $[x_0, y_0]$ of two segments$[x_1, y_1]$, $[x_2, y_2]$ and $[x_3, y_3]$, $[x_4, y_4]$, given in parametric form (one of which is a fragment of the curve). This problem can be represented as a system consisting of two linear equations with unknown parameters of the first and second segments:

$$\begin{cases} x_0 = (x_2 - x_1)t_1 + x_1 = (x_4 - x_3)t_2 + x_3 \\ y_0 = (y_2 - y_1)t_2 + y_1 = (y_4 - y_3)t_2 + y_3 \end{cases},$$

provided that the result of the solution will satisfy the following two conditions: $0 \leq t_1 \leq 1$ and $0 \leq t_2 \leq 1$, Otherwise, the segments are either parallel, or only the straight lines on which they lie intersect [3, 8].

Checklist Questions

1. What is the difference between the explicit and parametric representations of a line?
2. What are the advantages and / or disadvantages of various forms presented of conic section curves in computer graphics?
3. How to draw a Bezier curve using lines in parametric view?
4. What is the difference when constructing second- and third-order Bezier curves using parametric line segments?
5. How are affine transformations performed on Bezier curves?
6. Describe all the options for solving the problem of finding the intersection point of two segments, given in a parametric form.

Table 4.1

No.	Complexity	Assignments	Points
1	Basic	Setting an isotropic coordinate system for a resizable window	1
2		Output of conic section curves in according the variant of the task	2
3		Drawing a line and calculating its intersection points with a conic section curve in according the variant	2
4	Advanced	Specifying the position of points the segment, in the graphic area using the mouse manipulator	2
5		Using OOP	1

Table 4.2

No.	Circle representation		Ellipse representation		Hyperbola representation		Parabola representation	
	explicit	parametric	explicit	parametric	explicit	parametric	explicit	parametric
1					+			++
2				++		+		
3	++			+				
4	+					++		
5		++			+			
6		++						+
7			+				++	
8						+		++
9	+							++
10						+	++	
11						++		+
12		+			++			
13	++					+		
14			+					++
15		+				++		
16					++			+
17		++		+				
18	++						+	
19				++			+	
20				+		++		
21		++				+		
22			++				+	
23						++	+	

End of Table 4.2

No.	Circle representation		Ellipse representation		Hyperbola representation		Parabola representation	
	explicit	parametric	explicit	parametric	explicit	parametric	explicit	parametric
24	+				++			
25				+			++	
26		+					++	
27			+		++			
28		+						++
29			++			+		
30					+		++	
31	+						++	
32				+	++			
33						+		++
34		+				++		
35			++					+
36	+			++				
37		++			+			
38		++					+	
39				++		+		
40	++					+		
41	++		+					
42		+	++					
43					+		++	
44				+				++
45	++				+			
46				+		++		
47			++		+			
48				++				+
49			+			++		
50	++							+
51					++		+	
52		+		++				
53			+				++	
54	+				++			
55	+		++					
56						++	+	
57		++	+					
58					++			+
59				++	+			
60			++				+	

Practical work № 5

QUADRICS. 3D AFFINE TRANSFORMATIONS

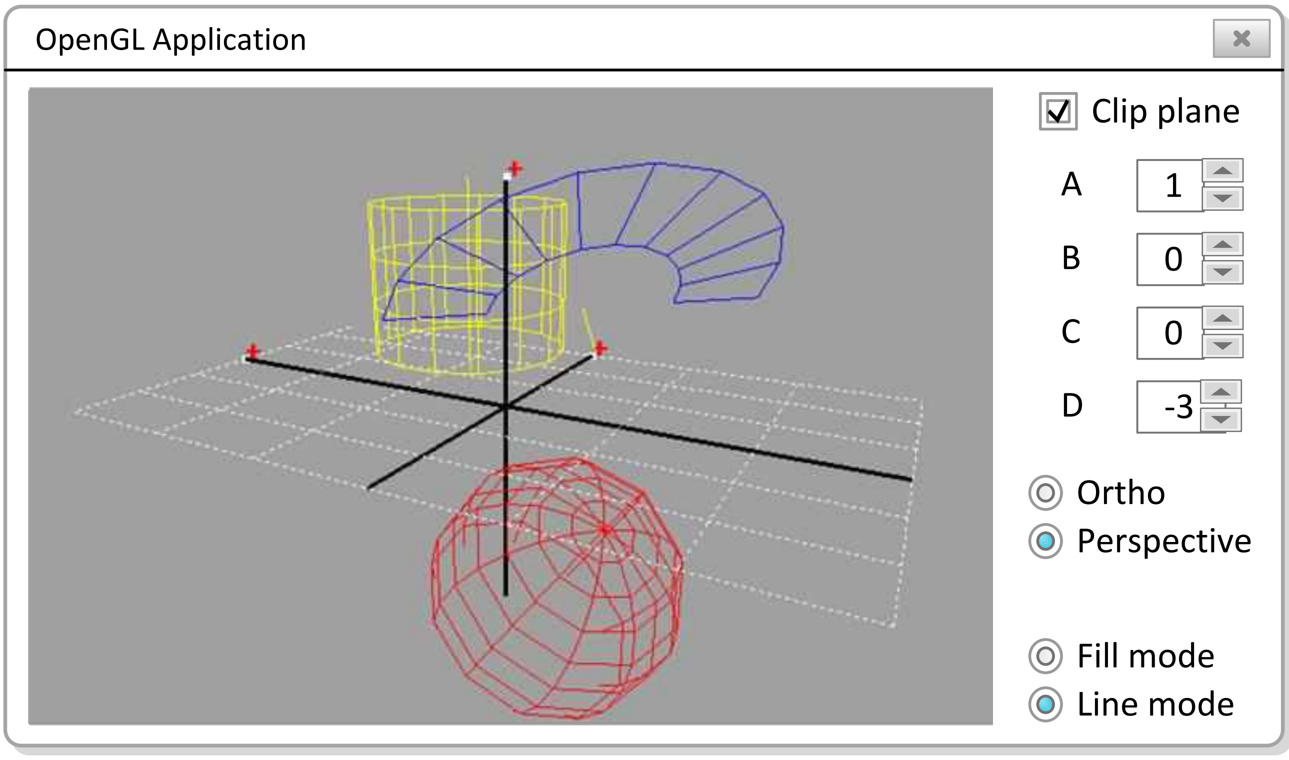
Aim of work: learn how to work with three-dimensional graphics primitives OpenGL and apply affine transformations to place objects in 3D space.

Task

Using the teacher specified development tools, develop a program using OpenGL tools that establishes an isotropic coordinate system, creates and displays an image of a three-dimensional scene with such elements (the assessment system is given in Table 5.1, and the options for the tasks are in Table 5.2):

- coordinate axes with zero in the center of the screen and indicate the axis and positive direction;
- coordinate grid (grid) in one of the planes (***X0Y***, ***X0Z*** or ***Y0Z***);
- three quadratic shapes – ***gluDisk*** / ***gluPartialDisk***, ***gluSphere***, ***gluCylinder*** in wireframe display mode and ***glEnable*** simplified lighting model (***GL_COLOR_MATERIAL***) for basic complexity;
- clipping plane for one of the shapes (sphere, cylinder, or cone);
- a full-fledged lighting model and / or textures for the implementation of a task with increased complexity.

The parameters of detailing objects (slices, stacks), color, thickness and line type are chosen independently. An example scene is shown in Fig. 5.1.



[✓] Clip plane
A 1
B 0
C 0
D -3
Ortho [✓ Perspective]
Fill mode [✓ Line mode]

Fig. 5.1. Example scene with quadratic objects and clipping plane for sphere

The minimum user interface should provide the ability to rotate the scene relative to the OX and OY axes using the mouse manipulator and control the clipping plane parameters [4].

Methodical instructions

An isotropic coordinate system can be established in two ways. In the first case, using the ***glViewport*** command, set the working area in accordance with the smaller value of the width / height of the window, and in the second, enter a correction factor (or divisor) equal to the ratio of the window width and height when setting the coordinate system (for example, with the ***glOrtho*** command). The depth must be set in such a way that, at any position of the given scene, all objects are within the visible area.

To display quadric primitives, an object origin point is used. In general, in the absence of affine transformations, this origin point of the object coincides with the zero point of the coordinate system. The origin point for the sphere and disk (full and partial) is their center, and for a cylinder (cone) – the center as one of the bases. To place each of the three quadrcs according to the variant of the task, you must use one of the affine transformations or their combination: rotation, translation, scaling.

After transformations, the origin point should be located at the coordinates $\boldsymbol{x_0}$, $\boldsymbol{y_0}$, $\boldsymbol{z_0}$, and the object axis should be parallel (‖) to a sphere / disk or collinear (⇈, ⇅) for a cylinder / cone, taking into account the option. When making images with quadric objects, the following parameters are used:

x_0, y_0, z_0	–	coordinates of the shape's origin point;
R	–	the radius of the sphere or the radius of the base of the cylinder / cone centered at the anchor point or the outer radius of the disc;
r	–	radius of the second base of the cylinder / truncated cone; inner radius of the disc;
h	–	cylinder / cone / truncated cone height;
∠start	–	the starting angle of the partial disc;
∠sweep	–	the end angle for partial disc;
axis ‖,⇈,⇅	–	parallelism / collinearity of coordinate axes and figures.

Checklist Questions

1. On what basis and by what command can the front and back sides of the surface be set?
2. What parameters of the lighting model are adjusted using the glLightModel command?
3. How does the surface normal affect the calculation of the illumination of objects and with what commands is it set?
4. What are the commands for setting fog in object lighting calculations?
5. What fog models does OpenGL use?

Table 5.1

No.	Complexity	Assignments	Points
1	Basic	Correct (isotropic) display of the task (when resizing the window) in orthographic projection	1
2		When launching the application, the axes ***0X***, ***0Y***, ***0Z***, are displayed, the grid and the wireframe of quadrac objects	1
3		Clipping plane parameters controled by user interface	1
4		Setting scene lights and material of quadrics with ***glColorMaterial*** command	1
5		Using ***Display Lists***	1
6	Advanced	Creating a perspective view of a scene	1
7		Blending a texture onto a surface with shapes specified by a variant	1
8		Using the ***glMaterial*** command to adjust the reflection parameters of the surface of scene objects	1

Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	$\measuredangle_{start}$	$\measuredangle_{sweep}$
1	Y0Z	sphere	∥ 0X	-1.5	-0.5	+2.5	1.0	-	-	-	-
		cone	⇅ 0Y	+3.5	+1.5	+4.0	1.5	0.0	1.5	-	-
		disk	∥ 0Z	+2.0	-2.0	-3.5	1.5	0.0	-	-	-
2	X0Y	sphere	∥ 0Y	-4.0	-2.5	-4.5	3.0	-	-	-	-
		truncated cone	⇅ 0X	-4.5	+2.5	+2.5	2.5	0.5	2.0	-	-
		disk	∥ 0Z	+3.5	-1.0	+3.0	2.0	0.0	-	-	-
3	X0Y	sphere	∥ 0X	+2.5	+3.5	+4.5	3.0	-	-	-	-
		truncated cone	⇈ 0Z	+1.5	-0.5	-1.5	1.0	2.0	1.5	-	-
		partial disk	∥ 0Y	-3.5	-2.5	+2.0	3.5	1.0	-	180°	90°
4	X0Y	sphere	∥ 0X	-2.5	-2.5	-1.5	2.0	-	-	-	-
		cylinder	⇈ 0Y	+1.5	-1.0	+3.0	1.5	-	2.0	-	-
		partial disk	∥ 0Z	-4.0	+2.5	+4.5	4.0	1.5	-	90°	90°
5	X0Y	sphere	∥ 0X	-2.0	+2.0	-3.0	2.5	-	-	-	-
		truncated cone	⇅ 0Y	-1.5	-1.0	+3.0	1.5	0.5	1.0	-	-
		disk	∥ 0Z	+1.5	+0.5	+1.5	1.0	0.0	-	-	-

Continuation of Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	$\measuredangle_{start}$	$\measuredangle_{sweep}$
6	X0Y	sphere	∥ 0Y	+3.0	+2.0	-3.5	2.5	-	-	-	-
		cylinder	⇅ 0Z	-4.5	-3.0	-3.5	3.0	-	1.5	-	-
		partial disk	∥ 0X	+4.5	-2.5	+1.5	2.5	1.0	-	270°	270°
7	X0Z	sphere	∥ 0Z	+2.5	-0.5	-2.5	1.0	-	-	-	-
		cone	⇅ 0Y	-3.5	-2.5	+2.5	0.0	2.5	2.0	-	-
		disk	∥ 0X	+4.5	+3.5	+4.5	4.0	1.5	-	-	-
8	X0Y	sphere	∥ 0X	-2.5	-1.0	-2.5	1.5	-	-	-	-
		cylinder	⇈ 0Y	-4.0	+2.5	+4.0	3.5	-	1.0	-	-
		disk	∥ 0Z	+3.5	-1.0	+3.0	1.0	0.0	-	-	-
9	X0Z	sphere	∥ 0Y	+1.5	+1.0	+2.5	1.5	-	-	-	-
		truncated cone	⇅ 0Z	-2.0	+1.5	-2.5	1.5	0.5	1.0	-	-
		partial disk	∥ 0X	+3.5	-0.5	-2.5	2.5	0.5	-	90°	45°
10	Y0Z	sphere	∥ 0X	-3.5	-3.0	-3.5	3.0	-	-	-	-
		truncated cone	⇅ 0Y	-2.0	+2.0	+3.5	2.0	0.5	2.0	-	-
		partial disk	∥ 0Z	+2.5	+2.5	-2.0	2.5	0.5	-	135°	90°
11	X0Z	sphere	∥ 0Y	-1.5	+1.0	+3.0	1.5	-	-	-	-
		cone	⇅ 0Z	+3.5	-1.0	+3.0	0.0	3.0	2.5	-	-
		disk	∥ 0X	-4.0	-2.0	-2.5	3.0	1.0	-	-	-
12	X0Z	sphere	∥ 0Z	+4.0	+1.5	+2.5	2.0	-	-	-	-
		cylinder	⇈ 0X	-3.5	+1.5	-4.5	1.5	-	3.0	-	-
		disk	∥ 0Y	+3.5	-1.0	-3.5	3.0	1.5	-	-	-
13	Y0Z	sphere	∥ 0Z	-4.5	-3.5	+3.5	3.5	-	-	-	-
		truncated cone	⇅ 0Y	+3.0	+2.0	+2.5	2.5	1.0	2.0	-	-
		partial disk	∥ 0X	+4.5	-3.5	-4.5	4.5	1.5	-	180°	90°
14	X0Z	sphere	∥ 0Z	+2.5	+1.0	-2.5	1.5	-	-	-	-
		truncated cone	⇈ 0Y	-3.0	-2.5	-4.5	3.0	1.0	2.5	-	-
		partial disk	∥ 0X	-3.5	+0.5	+2.0	2.0	0.0	-	225°	180°
15	X0Y	sphere	∥ 0Z	-2.5	-2.5	-2.5	2.5	-	-	-	-
		cone	⇈ 0Y	-4.0	+2.0	+2.5	0.0	2.5	1.5	-	-
		partial disk	∥ 0X	+1.5	-1.5	+3.5	3.5	0.5	-	135°	270°
16	X0Z	sphere	∥ 0Z	+4.0	-1.5	+2.0	2.0	-	-	-	-
		cone	⇈ 0X	-3.5	+0.5	+2.0	0.0	1.0	2.5	-	-
		partial disk	∥ 0Y	-2.5	-1.5	-3.5	3.5	1.0	-	0°	45°

Continuation of Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	∡*start*	∡*sweep*
		sphere	‖ 0X	+2.0	+1.5	+2.0	2.0	-	-	-	-
17	X0Z	cylinder	⇅ 0Y	+4.5	-3.5	-4.0	4.0	-	1.0	-	-
		disk	‖ 0Z	-2.0	+2.0	-2.5	1.5	0.0	-	-	-
		sphere	‖ 0Y	-2.0	-2.5	-4.0	2.5	-	-	-	-
18	X0Z	cylinder	⇈ 0X	+2.0	-2.5	+4.5	2.0	-	1.0	-	-
		disk	‖ 0Z	+3.5	+3.5	-3.5	3.0	0.5	-	-	-
		sphere	‖ 0Y	+2.5	+1.0	-3.5	1.5	-	-	-	-
19	X0Z	cylinder	⇅ 0X	-2.5	-3.5	-4.5	2.5	-	1.5	-	-
		partial disk	‖ 0Z	-3.5	+3.0	+3.0	3.5	1.0	-	180°	135°
		sphere	‖ 0X	-1.5	+1.5	+3.5	2.0	-	-	-	-
20	X0Z	cone	⇈ 0Y	+3.5	-3.0	+3.0	3.0	0.0	2.5	-	-
		partial disk	‖ 0Z	-2.0	-2.5	-4.0	2.5	0.5	-	180°	90°
		sphere	‖ 0Y	+2.5	+0.5	+2.0	1.0	-	-	-	-
21	Y0Z	cylinder	⇈ 0Z	-3.5	+3.5	-4.5	3.9	-	3.5	-	-
		disk	‖ 0X	-3.5	-1.0	+3.5	3.0	1.0	-	-	-
		sphere	‖ 0X	-1.5	-0.5	+1.5	1.0	-	-	-	-
22	X0Z	cone	⇅ 0Z	+3.0	+1.5	+1.5	2.5	0.0	1.5	-	-
		disk	‖ 0Y	-4.0	+2.5	-3.5	3.0	1.0	-	-	-
		sphere	‖ 0Y	-4.5	+3.0	-3.0	3.0	-	-	-	-
23	X0Y	cone	⇈ 0Z	-2.5	-1.0	+2.5	0.0	2.0	2.0	-	-
		partial disk	‖ 0X	+2.5	+3.5	+4.0	4.0	1.5	-	90°	180°
		sphere	‖ 0Z	+2.0	+2.0	+3.5	2.0	-	-	-	-
24	Y0Z	truncated cone	⇅ 0Y	-3.5	+2.5	-2.0	2.0	0.5	2.0	-	-
		disk	‖ 0X	-2.0	-2.5	+3.5	2.5	0.5	-	-	-
		sphere	‖ 0X	+2.5	-0.5	-1.5	1.0	-	-	-	-
25	Y0Z	cone	⇈ 0Z	-2.0	+2.5	-3.5	2.0	0.0	3.0	-	-
		partial disk	‖ 0Y	+4.5	+2.5	+2.5	4.0	1.0	-	0°	45°
		sphere	‖ 0Z	+4.5	-3.0	+2.5	3.0	-	-	-	-
26	Y0Z	truncated cone	⇈ 0X	-2.5	-3.0	-3.0	3.0	0.5	2.0	-	-
		partial disk	‖ 0Y	-4.0	+2.5	+4.0	3.5	1.0	-	45°	90°
		sphere	‖ 0Z	-2.5	+3.0	+3.0	2.5	-	-	-	-
27	X0Y	cone	⇅ 0X	+3.5	+1.0	-3.5	2.0	0.0	1.0	-	-
		disk	‖ 0Y	-3.5	-0.5	-2.5	2.5	0.0	-	-	-

Continuation of Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	∡*start*	∡*sweep*
28	X0Z	sphere	∥ 0Y	+4.0	-2.5	-4.5	3.0	-	-	-	-
		cone	⇈ 0X	-1.5	-1.5	+4.5	1.5	0.0	2.5	-	-
		partial disk	∥ 0Z	+4.0	+2.5	+3.5	3.5	1.0	-	225°	225°
29	Y0Z	sphere	∥ 0Z	-2.5	+3.5	+4.5	3.0	-	-	-	-
		cylinder	⇈ 0Y	-2.5	-2.5	-1.5	1.5	-	2.5	-	-
		partial disk	∥ 0X	+2.0	-2.0	+2.5	2.5	0.5	-	135°	135°
30	Y0Z	sphere	∥ 0Z	+3.5	-2.5	+2.0	2.5	-	-	-	-
		cylinder	⇅ 0X	-3.5	-3.5	-3.5	3.0	-	1.0	-	-
		disk	∥ 0Y	-3.0	+2.0	+2.5	2.5	0.0	-	-	-
31	X0Y	sphere	∥ 0X	-4.0	+2.5	+4.0	3.0	-	-	-	-
		cone	⇅ 0Z	+4.0	+2.5	-4.5	3.0	0.0	3.5	-	-
		partial disk	∥ 0Y	-2.5	-1.0	-2.5	2.5	0.5	-	90°	45°
32	Y0Z	sphere	∥ 0X	+3.5	-0.5	+2.5	1.0	-	-	-	-
		truncated cone	⇈ 0Y	+4.0	+1.5	-1.5	1.5	2.0	1.5	-	-
		disk	∥ 0Z	-4.0	-2.5	-3.5	3.0	0.5	-	-	-
33	X0Y	sphere	∥ 0Y	-4.0	+2.0	-3.0	2.5	-	-	-	-
		cylinder	⇈ 0X	+2.5	+2.5	+2.0	2.5	-	1.0	-	-
		partial disk	∥ 0Z	+1.5	-0.5	-1.5	1.5	0.0	-	270°	135°
34	X0Y	sphere	∥ 0Z	+4.0	-2.0	-3.0	2.5	-	-	-	-
		cylinder	⇅ 0Y	+2.5	+3.0	+3.0	2.5	-	2.5	-	-
		partial disk	∥ 0X	-2.5	+3.5	-3.5	3.5	1.5	-	315°	225°
35	Y0Z	sphere	∥ 0Y	-4.0	-1.5	+2.5	2.0	-	-	-	-
		cylinder	⇈ 0Z	+2.0	+2.0	+2.5	2.0	-	1.0	-	-
		partial disk	∥ 0X	+4.0	-2.0	-2.5	2.5	1.0	-	225°	315°
36	Y0Z	sphere	∥ 0Z	+4.0	-2.5	-4.5	3.0	-	-	-	-
		cone	⇈ 0X	-2.5	+3.5	-4.0	0.0	2.0	2.0	-	-
		disk	∥ 0Y	+4.5	+3.0	+3.0	3.5	1.5	-	-	-
37	X0Y	sphere	∥ 0Z	+4.0	+2.0	+3.5	2.0	-	-	-	-
		truncated cone	⇈ 0X	+4.5	-3.0	-3.5	3.0	0.5	4.0	-	-
		disk	∥ 0Y	-4.0	-2.0	+3.0	2.5	0.5	-	-	-
38	X0Z	sphere	∥ 0Z	-2.5	-1.0	+2.5	1.5	-	-	-	-
		cylinder	⇅ 0X	+3.0	+2.5	+4.0	2.5	-	3.0	-	-
		partial disk	∥ 0Y	-4.5	+3.5	-3.5	4.0	1.0	-	135°	315°

Continuation of Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	$\measuredangle_{start}$	$\measuredangle_{sweep}$
39	X0Y	sphere	∥ 0X	+1.5	+0.5	-1.5	1.0	-	-	-	-
		cone	⇈ 0Z	-2.0	-2.0	-3.5	0.0	2.0	3.0	-	-
		disk	∥ 0Y	+4.5	-2.5	+2.5	3.0	0.5	-	-	-
40	X0Z	sphere	∥ 0Z	-4.5	-2.5	-2.5	2.5	-	-	-	-
		truncated cone	⇈ 0Y	+4.5	-3.0	+3.5	3.0	1.5	2.5	-	-
		disk	∥ 0X	+2.0	+2.5	-3.5	3.0	1.0	-	-	-
41	X0Y	sphere	∥ 0Z	-3.5	+3.5	-3.5	3.5	-	-	-	-
		truncated cone	⇅ 0X	+2.0	+1.5	+1.5	1.5	2.0	1.0	-	-
		partial disk	∥ 0Y	+2.5	-1.5	-3.5	3.5	1.0	-	90°	270°
42	X0Y	sphere	∥ 0Y	-3.5	-2.5	-2.0	2.5	-	-	-	-
		truncated cone	⇅ 0X	+2.5	-1.0	+3.5	1.0	2.0	1.5	-	-
		partial disk	∥ 0Z	-4.5	+3.0	+3.5	4.5	2.0	-	45°	90°
43	X0Y	sphere	∥ 0Y	-3.5	-3.5	-3.5	4.0	-	-	-	-
		cylinder	⇅ 0Z	+3.5	-2.5	+2.0	2.5	-	2.0	-	-
		disk	∥ 0X	-2.5	+2.5	+2.0	2.5	0.5	-	-	-
44	Y0Z	sphere	∥ 0Y	-2.5	+1.0	-3.5	1.5	-	-	-	-
		truncated cone	⇈ 0X	+3.5	-3.5	-4.5	3.5	1.0	3.0	-	-
		disk	∥ 0Z	+4.5	+2.5	+2.0	3.0	0.5	-	-	-
45	X0Z	sphere	∥ 0Y	+2.0	+1.5	+2.0	2.0	-	-	-	-
		truncated cone	⇈ 0Z	+1.5	-1.5	-3.5	1.5	0.5	3.0	-	-
		disk	∥ 0X	-3.5	+0.5	-1.5	1.5	0.0	-	-	-
46	X0Z	sphere	∥ 0X	+3.5	-3.0	+3.5	3.5	-	-	-	-
		truncated cone	⇈ 0Z	-4.0	+1.5	+2.0	2.0	0.5	1.0	-	-
		disk	∥ 0Y	-3.0	-2.5	-4.0	3.0	0.5	-	-	-
47	X0Y	sphere	∥ 0Y	+2.5	-1.5	-3.5	2.0	-	-	-	-
		cone	⇈ 0Z	+3.5	+1.0	+3.0	3.0	0.0	2.5	-	-
		disk	∥ 0X	-2.5	+3.0	-2.5	2.0	0.0	-	-	-
48	X0Z	sphere	∥ 0X	+3.5	+1.5	-3.5	2.0	-	-	-	-
		cone	⇈ 0Y	-4.0	-2.5	-4.0	2.5	0.0	3.0	-	-
		disk	∥ 0Z	-4.5	+2.5	+1.5	2.5	0.0	-	-	-
49	Y0Z	sphere	∥ 0Y	-2.5	-3.5	-4.5	2.5	-	-	-	-
		truncated cone	⇅ 0Z	-2.5	+1.5	+4.0	2.0	0.5	3.0	-	-
		disk	∥ 0X	+2.5	+2.5	-2.5	2.0	0.5	-	-	-

End of Table 5.2

No.	*Grid*	Shape	Parameter values for quadrics								
			axis	x_0	y_0	z_0	R	r	h	$\measuredangle_{start}$	$\measuredangle_{sweep}$
50	X0Z	sphere	∥ 0X	-2.0	-2.5	-3.5	2.5	-	-	-	-
		truncated cone	⇅ 0Z	+3.5	-3.0	+2.5	3.0	1.0	2.0	-	-
		partial disk	∥ 0Y	+2.0	+1.5	-1.5	2.0	0.0	-	270°	270°
51	Y0Z	sphere	∥ 0X	+2.5	-3.5	+4.0	2.5	-	-	-	-
		cylinder	⇅ 0Z	-2.5	-0.5	-2.5	1.0	-	1.0	-	-
		partial disk	∥ 0Y	-2.5	+3.5	+4.5	4.5	1.5	-	180°	315°
52	Y0Z	sphere	∥ 0X	-3.5	-2.5	-1.5	2.0	-	-	-	-
		cylinder	⇈ 0Z	-2.0	+2.0	+2.5	2.0	-	1.0	-	-
		disk	∥ 0Y	+3.5	+0.5	-2.0	2.5	0.0	-	-	-
53	X0Z	sphere	∥ 0X	+3.5	+1.5	+4.0	2.0	-	-	-	-
		cylinder	⇅ 0Y	-3.0	+2.5	-3.5	3.0	-	2.5	-	-
		partial disk	∥ 0Z	+3.0	-1.5	-2.0	3.0	1.0	-	45°	45°
54	X0Y	sphere	∥ 0Z	-3.5	+3.5	-3.5	3.5	-	-	-	-
		cylinder	⇈ 0Y	+3.5	+3.5	+4.0	3.5	-	1.0	-	-
		disk	∥ 0X	+4.0	-2.5	-4.0	3.5	1.5	-	-	-
55	Y0Z	sphere	∥ 0Y	+3.5	-0.5	+2.5	1.5	-	-	-	-
		cone	⇅ 0X	+2.0	+2.5	-3.5	2.5	0.0	3.0	-	-
		partial disk	∥ 0Z	-2.5	-2.5	-2.5	2.5	0.5	-	90°	270°
56	Y0Z	sphere	∥ 0Y	-4.0	+2.0	+3.5	2.5	-	-	-	-
		cone	⇅ 0X	-2.5	-0.5	-1.5	1.5	0.0	3.0	-	-
		disk	∥ 0Z	+4.5	-3.5	+4.5	3.0	1.0	-	-	-
57	X0Z	sphere	∥ 0Z	+3.0	+2.0	+2.5	2.0	-	-	-	-
		cone	⇅ 0Y	+3.5	-3.5	-3.5	2.5	0.0	3.0	-	-
		partial disk	∥ 0X	-3.5	+3.5	-3.5	3.5	1.0	-	315°	90°
58	X0Y	sphere	∥ 0X	+4.5	+2.5	+2.0	2.5	-	-	-	-
		cylinder	⇅ 0Z	+4.5	-2.5	-2.5	2.5	-	1.0	-	-
		disk	∥ 0Y	-4.0	-2.5	+4.5	3.5	0.5	-	-	-
59	Y0Z	sphere	∥ 0Y	+4.5	+3.0	+2.5	3.0	-	-	-	-
		truncated cone	⇈ 0X	-1.5	+1.5	-3.5	1.5	2.5	1.5	-	-
		partial disk	∥ 0Z	-4.5	-3.0	+3.0	4.0	1.5	-	45°	90°
60	Y0Z	sphere	∥ 0Z	-1.5	+1.0	-3.0	1.5	-	-	-	-
		cone	⇅ 0X	+1.5	-0.5	-1.5	1.0	0.0	3.0	-	-
		partial disk	∥ 0Y	+4.5	+2.5	+1.5	4.0	1.5	-	135°	45°

Practical work № 6

SCREEN SAVER & ANIMATION

Aim of work: study of methods and ways of animation by using computational resources of operating system during equipment idle time.

Task

Using the teacher specified development tools, create a ScreenSaver with animation (Table 6.1). The general algorithm of such a program in Windows is shown in Fig. 6.1. Choose the animation plot yourself and agree with the teacher.

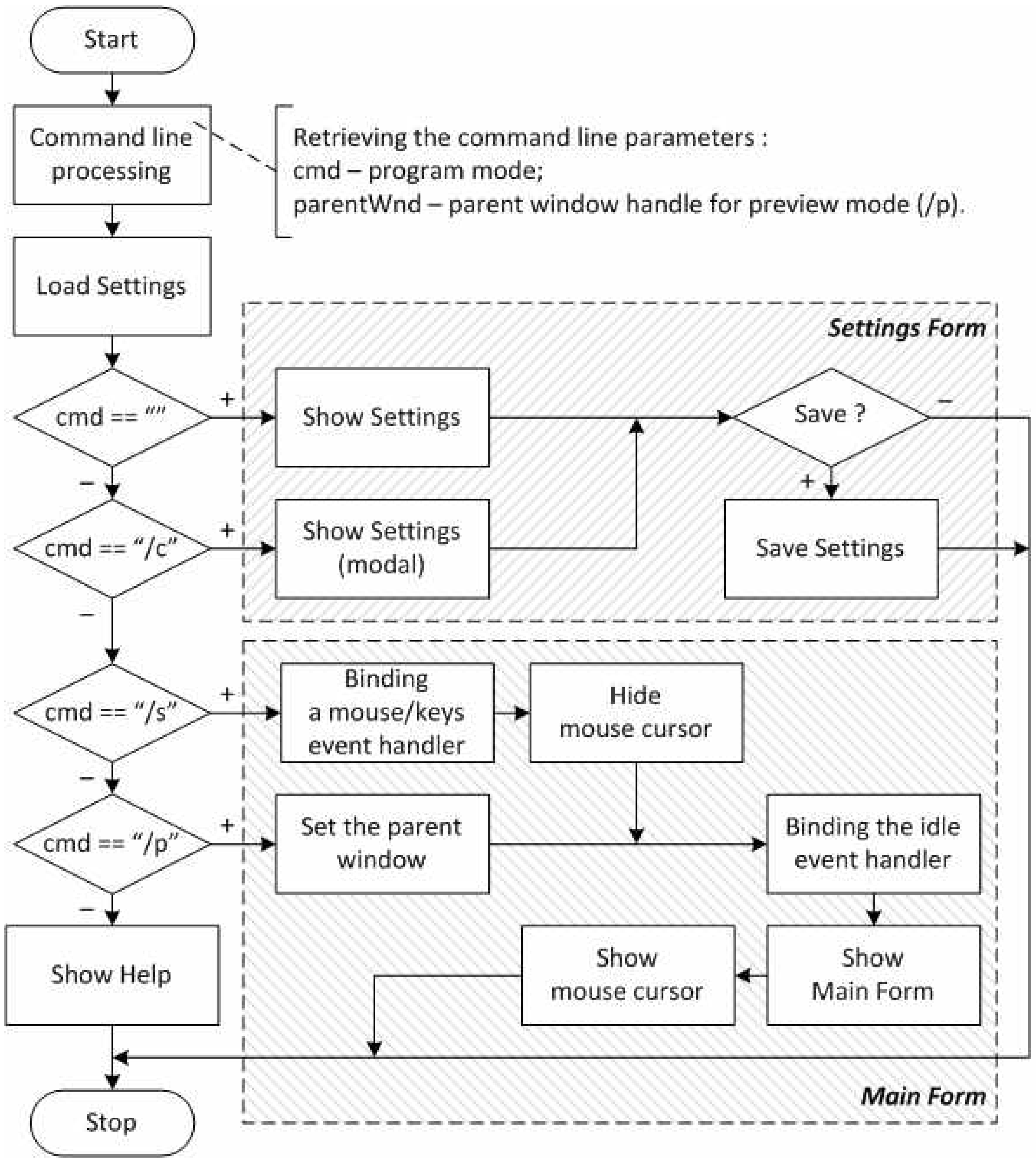


Fig. 6.1. The general algorithm of the screen saver in OS Windows

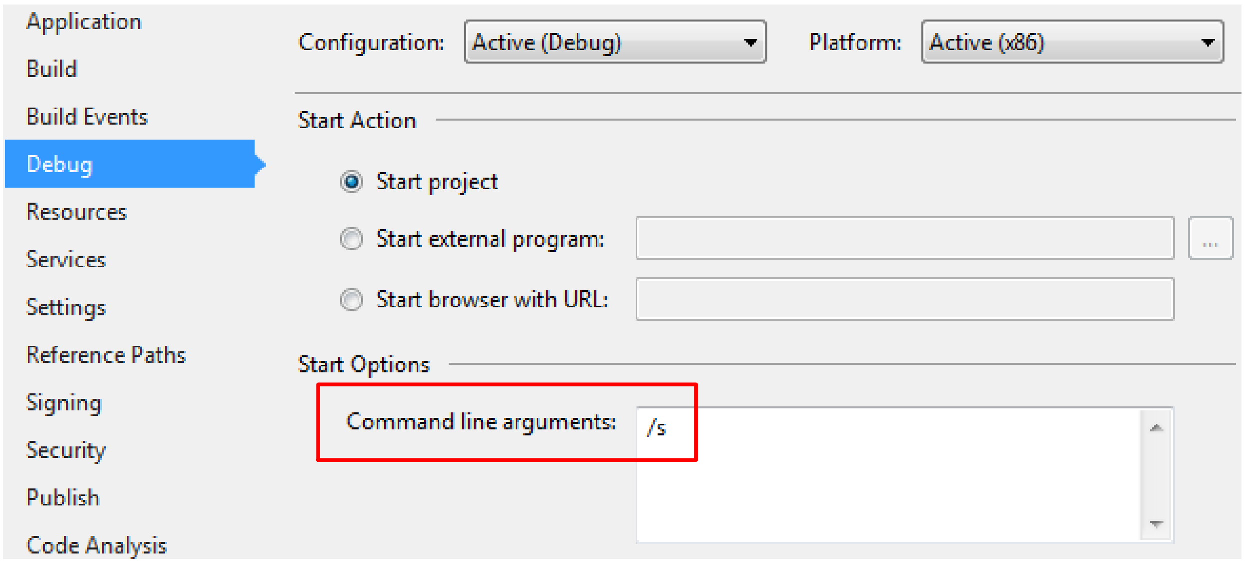
Methodical instructions

Actually ScreenSaver is a typical executable file (only with the .SCR extension instead of .EXE), which is controlled via command line parameters («/c» – *configure*, «/p» – *preview*, «/s» – *show*):

ScreenSaver.scr	–	show the settings window;
ScreenSaver.scr /c	–	show the settings window modally;
ScreenSaver.scr /s	–	main full-screen mode of operation;
ScreenSaver.scr /p hWnd	–	preview of the main mode in the parent window with the hWnd descriptor;
ScreenSaver.scr /a	–	setting a password in Windows 95 (legacy mode, currently not used).

Usually the run modes can be implemented separately in two window: Setting Form and Main Form (see Fig. 6.1), but only one of them works at application startup. To select a run mode for development and debugging (under Visual Studio control), the command line parameter is set in the project properties (Fig. 6.2).

By default, all screen savers are located in directory *«c:\windows\system32\*.scr»,* where you can place your program, too [9]. Also, available installation and manual test are running the screensver from the context menu of the operating system (Fig. 6.3).



Configuration: Active (Debug)

Platform: Active (x86)

Command line arguments: /s

Fig. 6.2. Setting command line parameter in project properties

It is recommended to divide the development process into several steps. At the first step, the analysis of the command line is carried out. At the second, a dialog box with settings and a mechanism for reading / saving them if changed. At the third, a window with graphics output and animation binding to the program inactivity mechanism carried out [11, 12]. After that develop interruption running at idle time in the main window by event from the manipulator «mouse» or keyboard and adapt algorithm of the ***Main Form*** for working in the parent window for the preview mode.

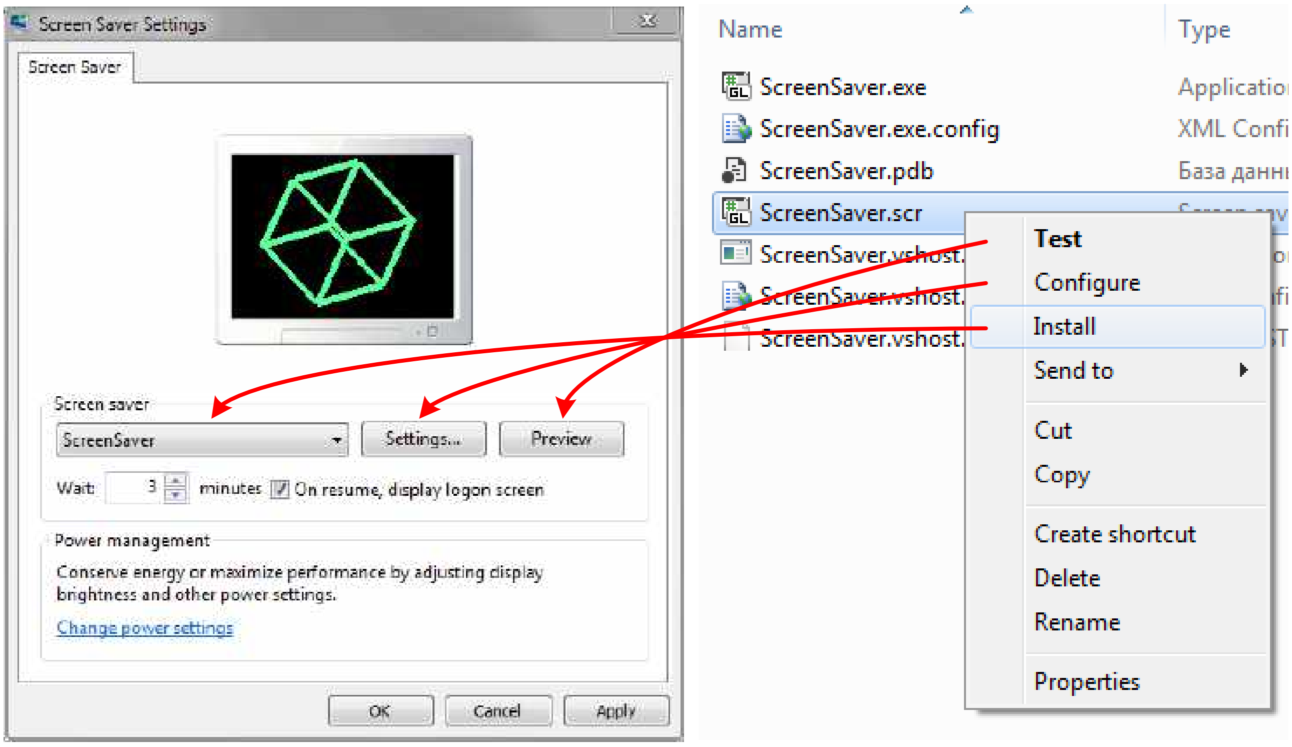


Fig. 6.3. Installing, starting and configuring the Screen Saver in OS Windows

Checklist Questions

1. When is double / triple buffering used?
2. How is double buffering implemented in OpenGL?
3. What methods and means of double buffering are there in WinAPI?
4. What are sprites, what are the principles of their use?
5. How to implement animation with minimal CPU usage?
6. What is meant by the terms parent / child window?

Table 6.1

No.	Complexity	Assignments	Points
1	Basic	Binding animation to operating system idle mechanism	2
2		Full screen implementation (command "/ s")	1
3		Implementing Screen Saver Settings ("/ c" Command)	1
4		Implementation of preview (command "/ p")	1
5	Advanced	Saving the ScreenSaver configuration and settings in the OS registry	1
6		Using complex and spectacular algorithms for image formation (for example, fractals)	2

Practical work № 7

FORWARD KINEMATICS VISUALIZATION

Aim of work: learn how to use affine transformations to create and control the model of physical objects.

Task

Using the teacher specified development tools, create an application for displaying the manipulator model specified by a kinematic scheme (the complexity estimate is given in Table 7.1). To control the model and the viewpoint it's necessary to use the keyboard and / or the mouse manipulator to change the values of the model's parameters such as the angles $\boldsymbol{\varphi}$, $\boldsymbol{\theta}$, $\boldsymbol{\psi}$, the distance ***S*** (see variants in Table 7.2).

Methodical instructions

To simplify calculations, the problem is first considered on the ***XOY*** plane. After that adding rotation around the ***OY*** axis and scaling to the 2D model complete 3D model is obtained.

To solve the problem on the plane, a combination of two affine transformations rotation and translation is required. These transformations correspond to the OpenGL commands:

$$Rz(\varphi) = \begin{vmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \rightarrow$$

```
// clockwise
glRotate[f,d]( φ,0,0,-1);
```

$$T(\Delta y) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \rightarrow$$

```
glTranslate[f,d](0, Δy,0).
```

The segment to convert is also specified by the matrix and the *OpenGL* code:

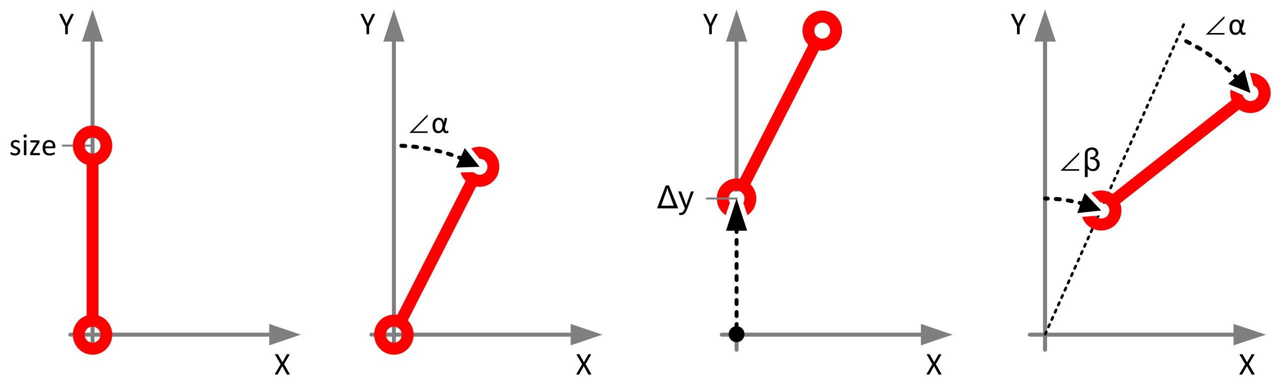
$$Segment(size) = \begin{vmatrix} 0 & 0 \\ 0 & size \\ 0 & 0 \\ 0 & 0 \end{vmatrix} \rightarrow$$

```
glBegin(GL_LINES);
   glVertex3d(0.0, 0.0, 0.0);
   glVertex3d(0.0, size, 0.0);
glEnd();
```

Thus, you should select your set of affin transformations for each segment of the manipulator step by step. For example, the sequence of transformations (Fig. 7.1) corresponds to the expression:

$$Sx(size, \alpha, \beta, \Delta y) = Rz(\beta) \times T(\Delta y) \times Rz(\alpha) \times Segment(size).$$

In addition, some articulation points and a slip point form a triangle, the angles of which depend on the value of the parameter ***S*** and are calculated based on the cosine theorem (see Appendix 3).



1 – $Segment(size)$ 2 – $rotate\, Rz(\alpha)$ 3 – $translate\, T(\Delta y)$ 4 – $rotate\, Rz(\beta)$

Fig. 7.1. Phased formation of a segment in the kinematic scheme

Table 7.1

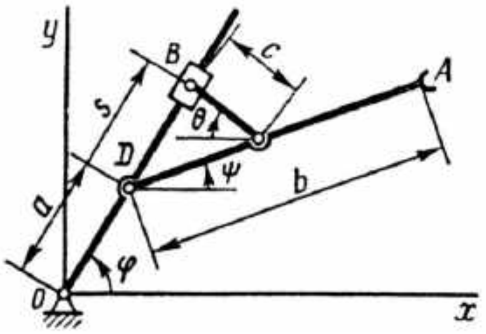
No.	Complexity	Assignments	Points
1	Basic	Implementation of the program of the two-dimensional model of the manipulator in accordance with the option	5
2		Modification of the program to a three-dimensional model (rotation of the observation point, scale)	1
3		Model and point of view control with mouse and / or keyboard	1
4		Using quadratic primitives to display a kinematic diagram	3
5		Using lighting and defining materials with glColorMaterial (...)	4
6		Report according to the design example	6
7	Advanced	Using OOP (developing your own classes)	1
8		Using textures for kinematic elements	2
9		Defining materials with glMaterial (...), using transparency	2
10		Using perspective projection to display the manipulator model	1
11		Implementing shadow lighting from the manipulator model	6

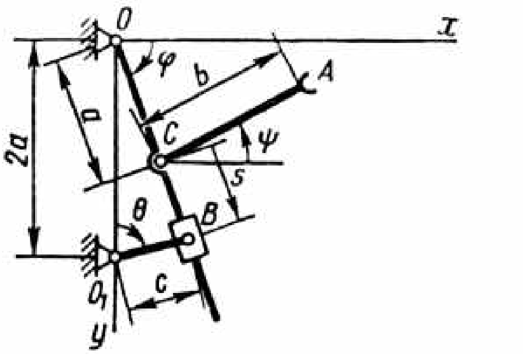
Checklist Questions

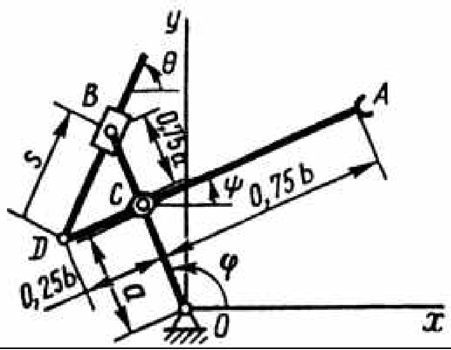
1. What data structures are needed to represent geometric objects in 3D space?
2. How is the stack of transformation matrices used in OpenGL?
3. How are light sources used in OpenGL scenes?
4. How are materials specified for OpenGL objects?

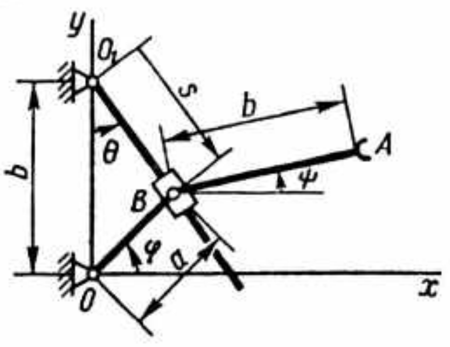
Table 7.2

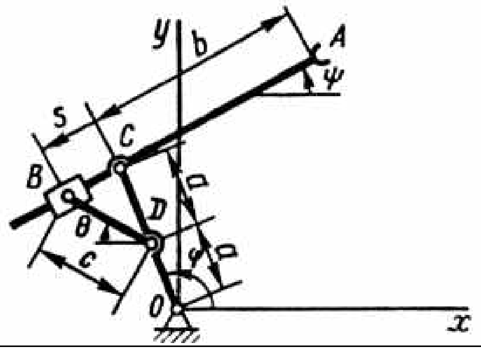
No.	Parameters	Kinematics model

1	a = 0.5
	b = 1.2
	c = 0.4
		 y
		B
		c
		A
		s
		θ
		D
		ψ
		b
		a
		φ
		0
		x

2	a = 0.4
	b = 0.7
	c = 0.24
		 0
		x
		φ
		b
		A
		a
		2a
		C
		ψ
		s
		θ
		B
		O1
		c
		y

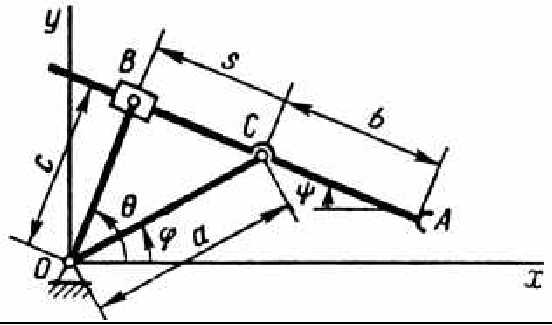
3	a = 0.4
	b = 1.2
		 y
		θ
		B
		A
		0,75a
		s
		C
		ψ
		0,75 b
		D
		0,25b
		a
		φ
		0
		x

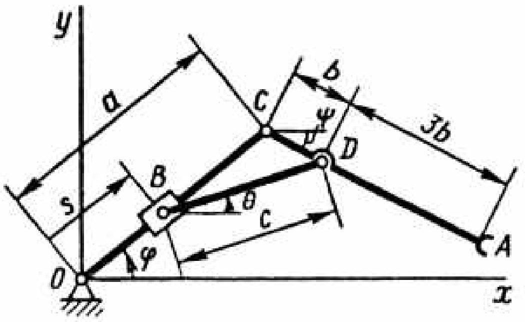
4	a = 0.4
	b = 0.7
		 y
		O1
		s
		b
		A
		b
		θ
		ψ
		B
		φ
		a
		0
		x

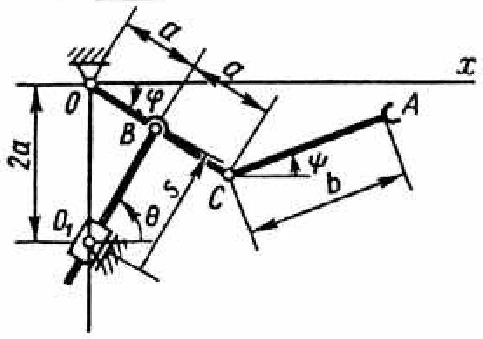
5	a = 0.3
	b = 0.8
	c = 0.4
		 y
		b
		A
		ψ
		s
		C
		a
		B
		D
		θ
		a
		φ
		c
		0
		x

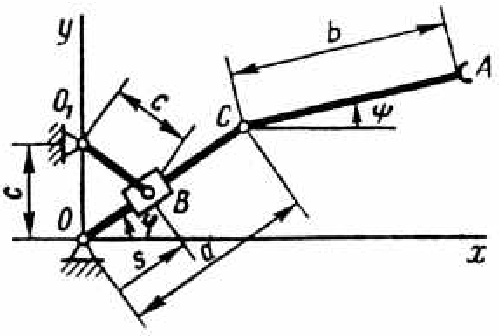
Continuation of Table 7.2

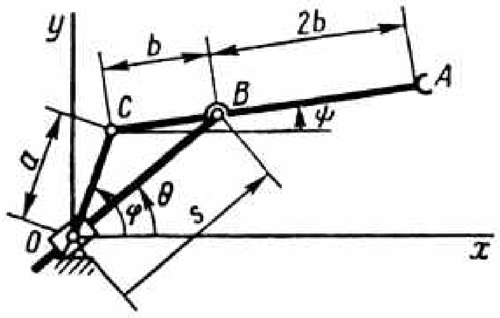
No.	Parameters	Kinematics model

6	a = 0.8
	b = 0.6
	c = 0.6
		 y
		B
		s
		C
		b
		c
		θ
		ψ
		φ
		a
		A
		O
		x

7	a = 0.9
	b = 0.24
	c = 0.64
		 y
		a
		C
		b
		ψ
		3b
		D
		B
		s
		θ
		c
		φ
		O
		A
		x

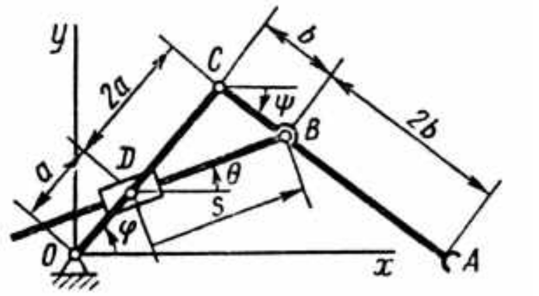
8	a = 0.3
	b = 0.66
		 a
		a
		x
		O
		φ
		A
		2a
		B
		s
		C
		ψ
		b
		O1
		θ

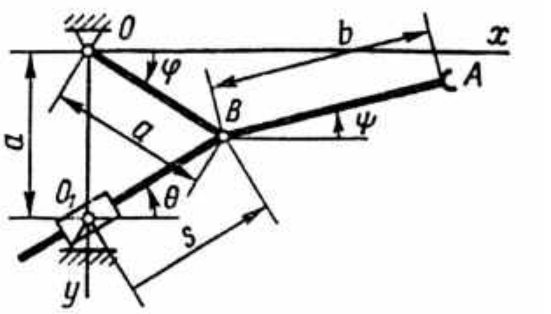
9	a = 0.74
	b = 0.8
	c = 0.32
		 y
		b
		A
		O1
		c
		C
		ψ
		c
		B
		O
		φ
		s
		a
		x

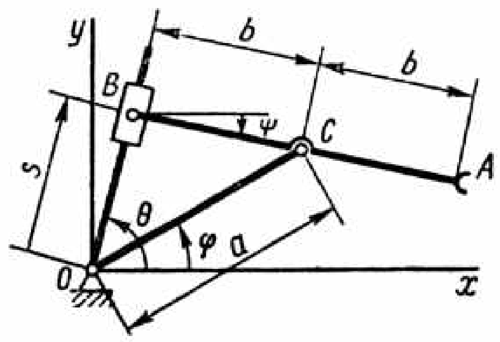
10	a = 0.3
	b = 0.36
		 y
		b
		2b
		A
		C
		B
		ψ
		a
		φ
		θ
		s
		O
		x

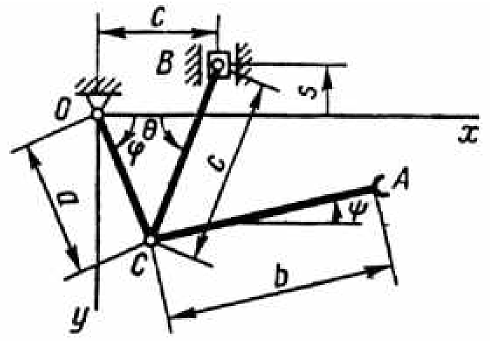
Continuation of Table 7.2

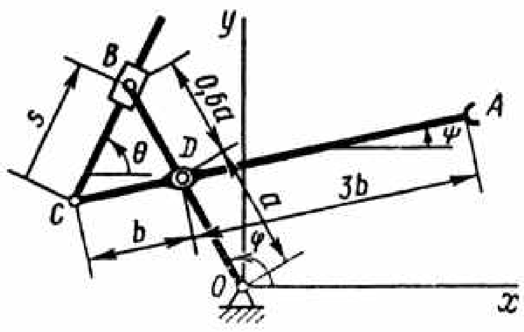
No.	Parameters	Kinematics model

11	a = 0.28
	b = 0.34
		 y
		2a
		a
		C
		b
		2b
		ψ
		B
		D
		θ
		s
		φ
		O
		x
		A

12	a = 0.6
	b = 0.8
		 O
		b
		x
		φ
		A
		B
		ψ
		a
		a
		O1
		θ
		s
		y

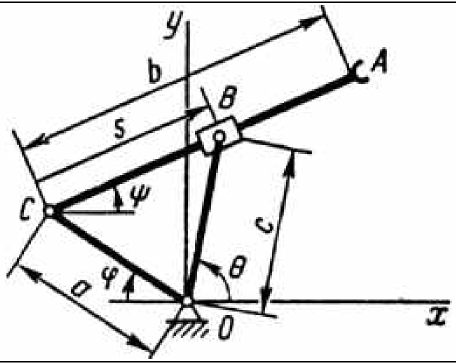
13	a = 0.8
	b = 0.6
		 y
		b
		b
		B
		ψ
		C
		A
		s
		θ
		φ
		a
		O
		x

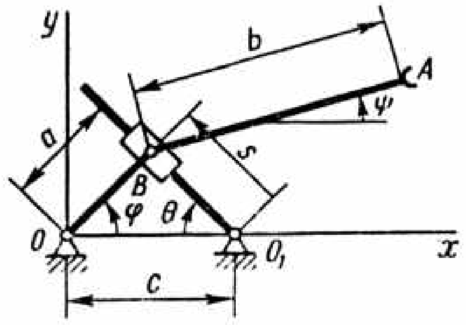
14	a = 0.46
	b = 0.82
	c = 0.6
		 c
		B
		s
		O
		θ
		φ
		x
		a
		c
		A
		ψ
		C
		b
		y

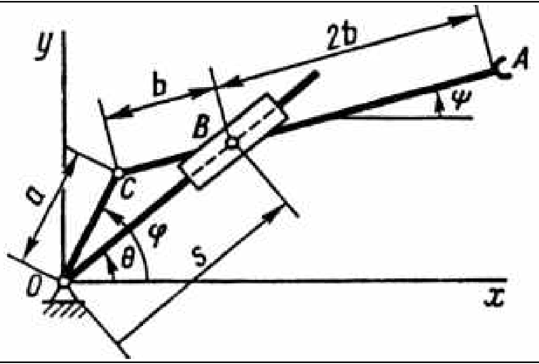
15	a = 0.5
	b = 0.3
		 y
		B
		0,6a
		A
		s
		θ
		D
		ψ
		C
		a
		3b
		b
		φ
		O
		x

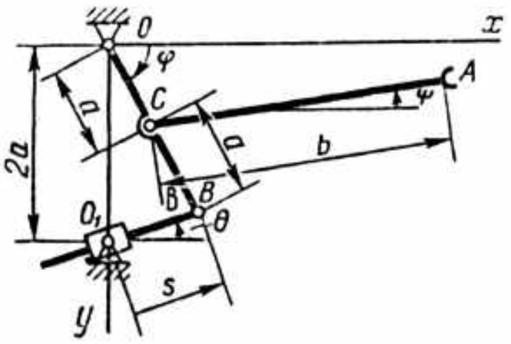
Continuation of Table 7.2

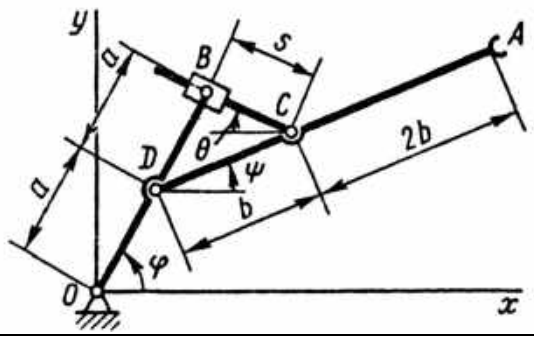
No.	Parameters	Kinematics model

16	a = 0.6
	b = 1.2
	c = 0.54
		 y
		b
		A
		B
		s
		C
		ψ
		c
		a
		φ
		θ
		O
		x

17	a = 0.42
	b = 1.0
	c = 0.56
		 y
		b
		A
		ψ
		a
		s
		B
		φ
		θ
		O
		O₁
		x
		c

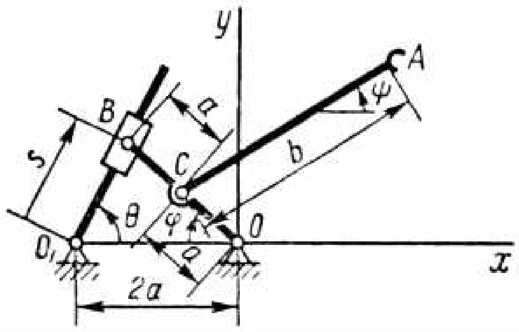
18	a = 0.4
	b = 0.46
		 y
		2b
		A
		b
		ψ
		B
		a
		C
		φ
		θ
		s
		O
		x

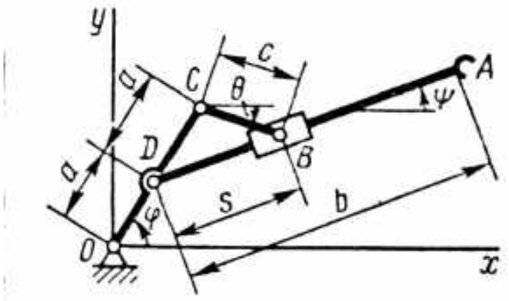
19	a = 0.34
	b = 0.9
		 O
		x
		φ
		A
		a
		C
		ψ
		2a
		a
		b
		B
		B
		O₁
		θ
		s
		y

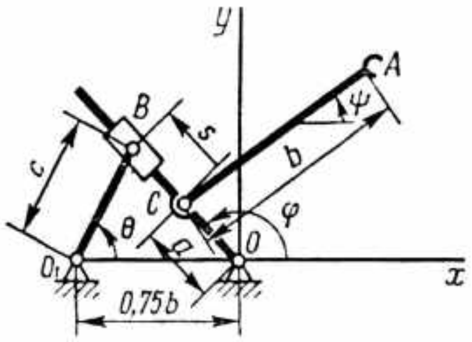
20	a = 0.4
	b = 0.5
		 y
		a
		B
		s
		A
		C
		D
		θ
		ψ
		2b
		a
		b
		φ
		O
		x

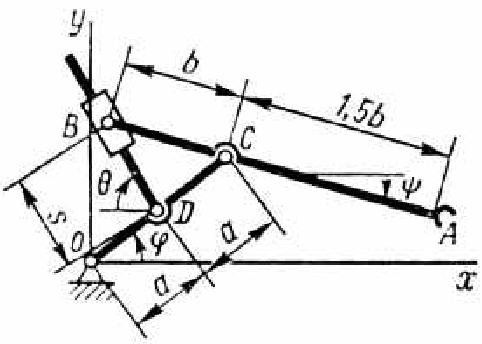
Continuation of Table 7.2

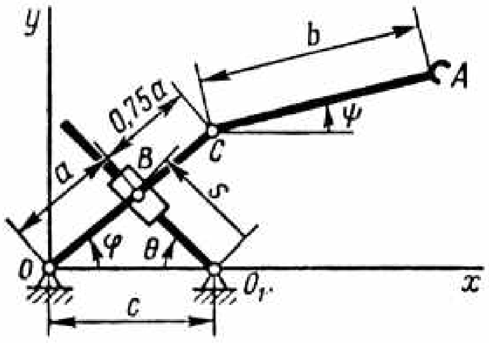
No.	Parameters	Kinematics model

21	a = 0.3
	b = 0.9
		 y
		A
		ψ
		B
		a
		s
		C
		b
		θ
		φ
		O₁
		O
		a
		2a
		x

22	a = 0.3
	b = 1.3
	c = 0.34
		 y
		c
		A
		C
		θ
		a
		ψ
		D
		B
		a
		S
		b
		φ
		O
		x

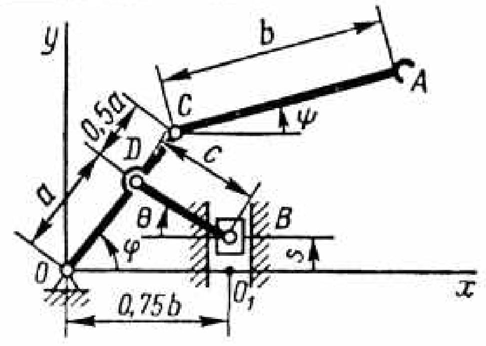
23	a = 0.24
	b = 0.82
	c = 0.4
		 y
		A
		B
		ψ
		s
		c
		b
		C
		φ
		θ
		a
		O
		O₁
		0,75b
		x

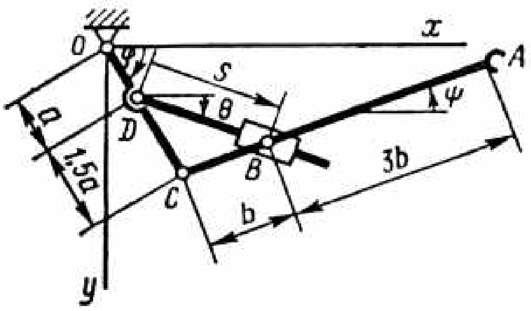
24	a = 0.3
	b = 0.5
		 y
		b
		1,5b
		B
		C
		θ
		ψ
		D
		s
		a
		A
		O
		φ
		a
		x

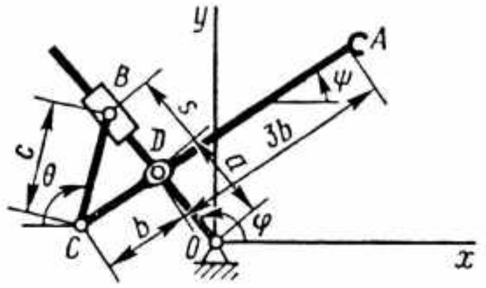
25	a = 0.4
	b = 0.9
	c = 0.6
		 y
		b
		A
		0,75a
		ψ
		C
		a
		B
		s
		φ
		θ
		O
		O₁
		c
		x

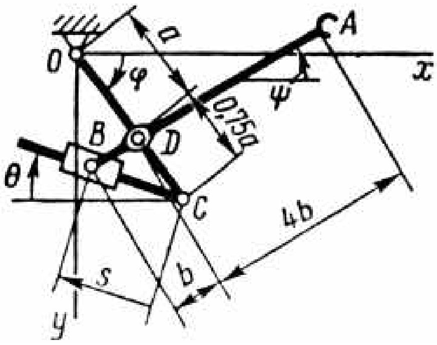
End of Table 7.2

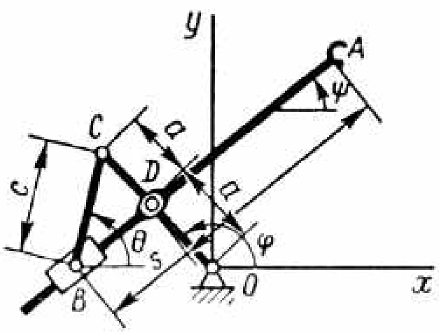
No.	Parameters	Kinematics model

26	A = 0.4
	b = 0.8
	c = 0.34
		 y
		b
		A
		C
		ψ
		0,5a
		D
		c
		a
		θ
		B
		φ
		s
		0
		0,75b
		O1
		x

27	a = 0.2
	b = 0.3
		 0
		x
		A
		φ
		S
		θ
		ψ
		a
		D
		1,5a
		B
		3b
		C
		b
		y

28	a = 0.3
	b = 0.3
	c = 0.4
		 y
		A
		B
		ψ
		s
		c
		3b
		D
		θ
		a
		φ
		C
		b
		0
		x

29	a = 0.34
	b = 0.24
		 a
		A
		0
		φ
		x
		ψ
		0,75a
		B
		D
		θ
		C
		4b
		s
		b
		y

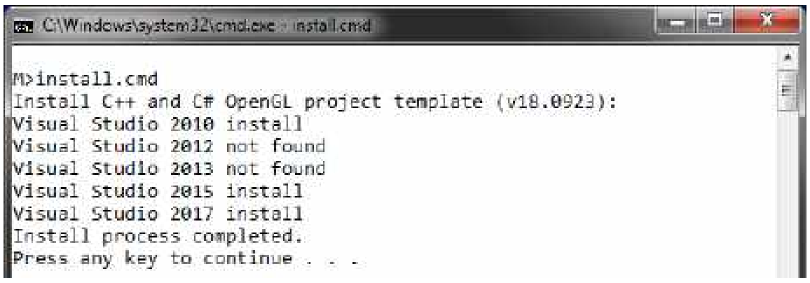
30	a = 0.28
	b = 0.86
	c = 0.4
		 y
		A
		ψ
		C
		a
		D
		c
		a
		θ
		s
		φ
		B
		0
		x

Appendix 1

INSTALLING PROJECT TEMPLATES FOR CREATING OPENGL APPLICATION IN C++ AND C#

1. Unzip the «***OpenGL projects templates.zip***» file.

2. If the operating system and ***Visual Studio*** are installed with the default settings, you can use the automatic installation using ***Install.cmd***, file, which copies the template files to the standard ***Visual Studio*** derictories (for 2010, 2012, 2013, 2015 and 2017 versions). As a result of the batch file operation, messages will be displayed on the screen, indicating for which versions of ***Visual Studio*** the project template files are installed (Fig. A.1.1).

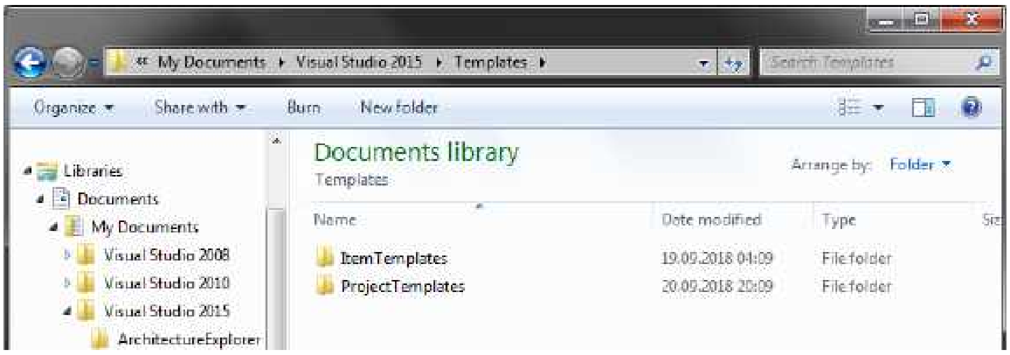


```
M>install.cmd
Install C++ and C# OpenGL project template (v18.0923):
Visual Studio 2010 install
Visual Studio 2012 not found
Visual Studio 2013 not found
Visual Studio 2015 install
Visual Studio 2017 install
Install process completed.
Press any key to continue . . .
```

Fig. A.1.1. Installing project templates for Visual Studio 2010, 2015 and 2017

3. If it was not possible to install templates in automatic mode, then it can be done manually by copying the ***ProjectTemplates*** and ***ItemTemplates*** directories from the *«**OpenGL projects templates.zip**»* archive according to the current system user's directories for each version of ***Visual Studio***, that requires these templates. For example, for ***Visual Studio 2015*** the project template directories are located as follows (Fig. A.1.2):

"***C:\Users\<User name>\Documents\Visual Studio 2015\Templates\...***"



Name	Date modified	Type
ItemTemplates	19.09.2018 04:09	File folder
ProjectTemplates	20.09.2018 20:09	File folder

Fig. A.1.2. Standard location of project template directories

4. When installing templates, it is recommended to close all running instances of ***Visual Studio*** or restart them after installation / copy project templates. Please note that installation directories may vary depending on system regional settings of ***Visual Studio*** versions and other user settings. Reinstallation overwrites the previous version of templates.

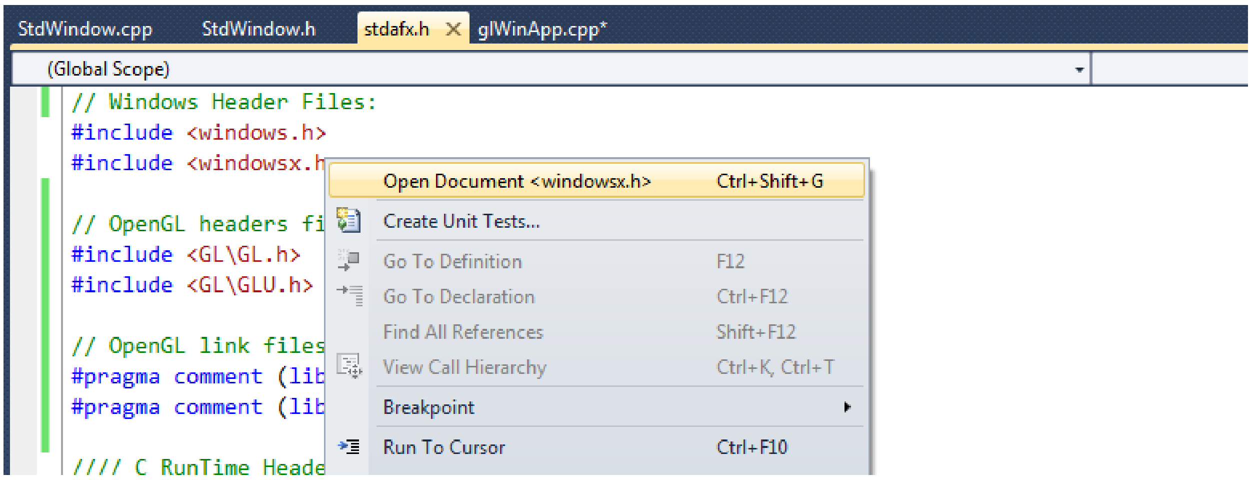
Appendix 2

USING WINDOWSX.H FILE IN C ++ PROJECTS TO HANDLE OPERATING SYSTEM MESSAGES

1. Find out which ***Windows*** message is responsible for the event requiring processing.

Example: the WM_SIZE message is responsible for the window resizing event.

2. Open the ***windowsx.h*** file (in the project it is included in the ***stdafx.h*** file, Fig. A.2.1) and search (***Ctrl + F***) for the line, which is formed as follows: in the message name, the prefix «WM_» is replaced with «On».

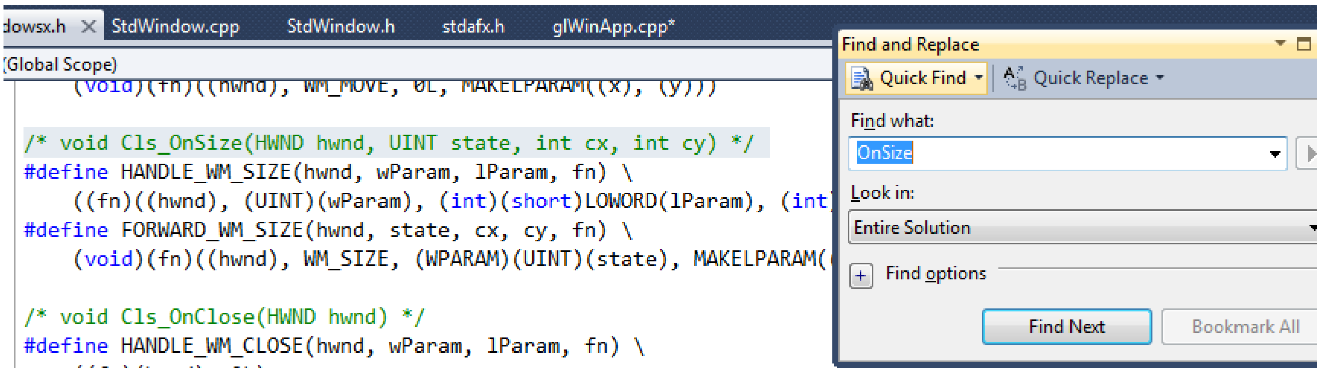


```
// Windows Header Files:
#include <windows.h>
#include <windowsx.h

// OpenGL headers fi
#include <GL\GL.h>
#include <GL\GLU.h>

// OpenGL link files
#pragma comment (lib
#pragma comment (lib

//// C RunTime Heade
```

Fig. A.2.1. Opening the windowsx.h file using the context menu

Example: The name of the message ***WM_SIZE*** is replaced with ***OnSize*** and the search for this string is performed (Fig. A.2.2). As a result, the commented out line with the prototype of the function that handles the ***WM_SIZE*** message is found:

```
/*void Cls_OnSize(HWND hwnd, UINT state, int cx, int cy)*/
```



```
    (void)(fn)((hwnd), WM_MOVE, 0L, MAKELPARAM((x), (y)))

/* void Cls_OnSize(HWND hwnd, UINT state, int cx, int cy) */
#define HANDLE_WM_SIZE(hwnd, wParam, lParam, fn) \
    ((fn)((hwnd), (UINT)(wParam), (int)(short)LOWORD(lParam), (int
#define FORWARD_WM_SIZE(hwnd, state, cx, cy, fn) \
    (void)(fn)((hwnd), WM_SIZE, (WPARAM)(UINT)(state), MAKELPARAM(

/* void Cls_OnClose(HWND hwnd) */
#define HANDLE_WM_CLOSE(hwnd, wParam, lParam, fn) \
```

Find what: OnSize

Look in: Entire Solution

Fig. A.2.2. Finding the prototype of the handler function in the ***windowsx.h*** file

3. Add the corresponding function to the main program (usually the ***Cls_*** prefix is removed) and execute the link between the message handling routine ***WndProc(...)*** and the event handler routine using the ***HANDLE_MSG(...)*** macro.

Example: based on the prototype found in the ***windowsx.h*** file, a function is added to the program code that will be responsible for the actions performed when the window is resized:

```
void OnSize(HWND hwnd, UINT state, int cx, int cy)
{
  // Commands executed when the window is resized
}
```

and binding this function to the general message processing algorithm of the operating system:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
  switch (message){
    HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
    HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);
    HANDLE_MSG(hWnd, WM_PAINT, OnPaint);
    HANDLE_MSG(hWnd, WM_SIZE, OnSize);
    default:
      return DefWindowProc(hWnd, message, wParam, lParam);
  }
}
```

4. Do the same with other messages of the operating system that require handling in application or non-standard response.

EXAMPLE OF FORWARD KINEMATICS VISUALIZATION TASK

A.3.1. Formulation of the problem

Develop an application for displaying a manipulator model with a given kinematic scheme (Fig. A.3.1) and relative dimensions ***a*** = 1.3, ***b*** = 1.1, ***c*** = 0.55, $\boldsymbol{a_1}$ = 0.31***a***, $\boldsymbol{a_2}$ = 0.69***a***). To control the movement of the manipulator and / or change the viewpoint, it is necessary to use the event handlers of the keyboard and / or the mouse manipulator, which change the value of the corresponding parameters:

– angle ***az*** – for rotation about the ***OZ*** axis;
– distance ***S*** – to change the angles ***at***, ***ag***;
– angle ***ay*** – for rotation about the ***OY*** axis.

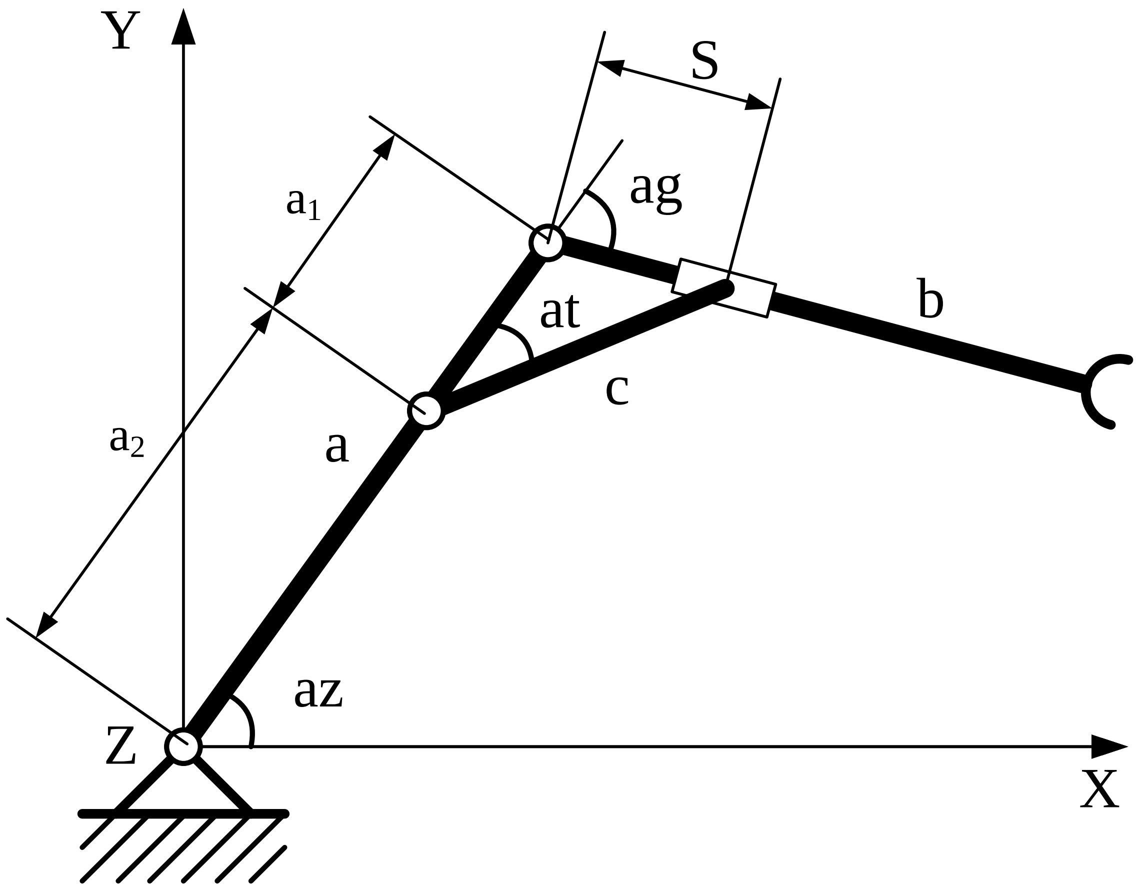


Fig. A.3.1 Kinematic scheme of the manipulator

A.3.2. Mathematical model of the kinematic scheme

The solution to the problem can be carried out in stages. At the first stage, the problem is considered as two-dimensional only in one plane. ***Z*** = ***const*** = 0 and at the second - in 3D space.

A.3.2.1. Affine transformations

To implement the task on the plane, you can use a combination of two affine transformations - rotation and translation. For the given variant, rotation about the ***Z*** axis is required:

$$Rz(\alpha) = \begin{vmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (A.3.1)$$

and translation along the ***Y*** axis:

$$Ty(\delta) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}. \qquad (A.3.2)$$

To implement the solution in space, additional rotation about the ***Y*** axis is required:

$$Ry(\alpha) = \begin{vmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}, \qquad (A.3.3)$$

rotation around the ***X*** axis:

$$Rx(\alpha) = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad (A.3.4)$$

and scaling along all three axes:

$$Mxyz(m) = \begin{vmatrix} m & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & m & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}. \qquad (A.3.5)$$

A.3.2.2. Decomposition by segments

In fact, the kinematic diagram can be represented in the form of three segments, which, for simplicity, can be represented as ***Segments*** of a given ***size***, which in the initial state is a vertical position (along the ***OY*** axis):

$$Segment(size) = \begin{vmatrix} 0 & 0 \\ 0 & size \\ 0 & 0 \\ 1 & 1 \end{vmatrix}. \qquad (A.3.6)$$

Thus, to solve the problem, it is necessary to determine a set of affine transformations, which is necessary for the transition from the initial position to the one given by the kinematic scheme.

Mathematical model of segment «a»

The location of segment ***a*** is the same as the starting position of all segments. To be displayed as part of a kinematic schema, we denote it as ***Sa***, depending on the rotation about the ***Z*** axis by the angle ***az***:

$$Sa(az) = Rz(az) \times Segment(a). \tag{A.3.7}$$

Mathematical model of segment «b»

The position of segment ***b*** depends on two angles – ***az*** and ***ag*** (see Fig. A.3.1). In this case, the angle ***ag***, depends on the value of ***S***. Therefore, to calculate the angle ***ag*** consider a triangle formed by the sides ***a₁, S*** and ***c***, in which one of the interior angles is adjacent to the angle ***ag***. Then we can use the cosine theorem, setting the angle adjacent to ***ag****,* as (***π – ag***) and connecting it by the following expression with the sides ***a₁, S*** and ***c***:

$$c^2 = a_1^2 + S^2 - 2a_1 S\cos(\pi - ag). \tag{A.3.8}$$

From here we can derive the dependence of the angle ***ag*** on the value of ***S***:

$$ag(S) = \pi - \arccos\left(\frac{a_1^2 + S^2 - c^2}{2a_1 S}\right). \tag{A.3.9}$$

Thus, to represent the segment as part of the kinematic schema, we introduce the designation ***Sb***, the position of which depends on the rotation about the ***Z*** axis by the angle ***az*** and the value of ***S***:

$$Sb(az, S) = Rz(az) \times Ty(a) \times Rz\big(ag(S)\big) \times Segment(b). \tag{A.3.10}$$

Mathematical model of segment «c»

The position of segment c depends on two angles – ***az*** and ***at*** (see Fig. A.3.1). In this case, the angle ***at***, depends on the value of ***S***. Therefore, to calculate the angle ***at*** consider a triangle formed by the sides ***a₁***, ***S***, ***c*** and containing the angle ***at***. Using the cosine theorem, we can associate the angle ***at*** with the sides ***a₁***, ***S*** and ***c***:

$$S^2 = a_1^2 + c^2 - 2a_1 c\cos(at) \tag{A.3.11}$$

and express its dependence on the value of ***S***:

$$at(S) = \arccos\left(\frac{a_1^2 + c^2 - S^2}{2a_1 c}\right). \tag{A.3.12}$$

Thus, to represent the segment as part of the kinematic schema, we introduce the identifier ***Sc***, the position of which depends on the rotation around the ***Z*** axis by the angle ***az*** and the value of ***S***:

$$Sc(az,S) = Rz(az) \times Ty(a_2) \times Rz\big(at(S)\big) \times Segment(c). \tag{A.3.13}$$

A.3.2.3. Physical limitations of the model

To calculate the position of the two segments, the cosine theorem is used, which connects the sides of the triangle ***a₁***, ***S*** and ***c***. Since the change in the value of ***S*** is used to control the kinematic scheme, it is necessary to take into account the limitation on the existence of the triangle: any side of the triangle is less than the sum of the other two sides. This constraint can be expressed by two predicates:

$$\begin{gathered} CheckMax(S) = S < a_1 + c; \\ CheckMin(S) = S > c - a_1, \end{gathered} \tag{A.3.14}$$

both of which are ***true*** when ***S*** is within acceptable limits.

A.3.2.4. General 3D mathematical model

Having considered the solution of the problem within one plane, we turn to the general three-dimensional model. To do this, we will add scaling and rotations about the ***X*** and ***Y*** axes to the rotation about the ***Z*** axis. All transformations performed for all segments will be denoted as ***Q***:

$$Q(ax,ay,az,m,S) = \ Rx(ax) \times Ry(ay) \times Mxyz(m) \times Rz(az), \tag{A.3.15}$$

then the full 3D model will look like this:

$$\begin{aligned} &Sa = Q(ax,ay,az,m,S) \times Segment(a); \\ &Sb = Q(ax,ay,az,m,S) \times Ty(a) \times Rz\big(ag(S)\big) \times Segment(b); \\ &Sc = Q(ax,ay,az,m,S) \times Ty(a_2) \times Rz\big(at(S)\big) \times Segment(c), \end{aligned} \tag{A.3.16}$$

and the transformations defined in ***Q***, for efficiency, can be stored, for example, in a matrix stack with subsequent restoration at the right moment.

A.3.3. Software implementation of the kinematic scheme

A.3.3.1. Source data

The source data of the kinematic scheme are set by the basic values ***a***, ***b***, ***c*** and can be represented in the program by ordinary variables. The auxiliary values ***a₁***, ***a₂*** are dependent on ***a***, so the direct assignment operation is prohibited for them, which is implemented by declaring them as read-only properties:

```
public partial class RenderControl : OpenGL
{
    public double a = 1.3, b = 1.1, c = 0.55;
    public double a1 { get { return 0.31 * a; } }
    public double a2 { get { return 0.69 * a; } }
...
}
```

Restrictions can be implemented in a similar way (A.3.14) to control the acceptable value of ***S***:

```
private double s = 0.2; // initial value
public double S
{
  get { return s; }
  set { if ((value >= (c-a1)) && (value <= a1+c)) s = value; }
}
```

The angles ***at*** and ***ag***, are also functional dependences on the value of ***S*** (A.3.9), (A.3.12), so the assignment operation is invalid for them. The ***arccos*** function returns the angle value in radians, and the *OpenGL* commands work with degrees, so we additionally perform the conversion from radians to degrees:

```
public double ag {
  get { return 180.0/Math.PI * // convert from radians to degrees
           (Math.PI - Math.Acos((a1*a1 + S*S - c*c) / (2*a1*S)));
    }
}

public double at {
  get { return 180.0/Math.PI * // convert from radians to degrees
           (Math.Acos((a1*a1 + c*c - S*S) / (2*a1*c)));
    }
}
```

A.3.3.2. Setup a coordinate system to viewport

For the correct display of the proportions of the elements of the kinematic scheme, it is necessary to use an isotropic coordinate system. This can be implemented in the window resize event handler as follows:

```
if (ClientSize.Width > ClientSize.Height){
    int dx = (ClientSize.Width - ClientSize.Height) / 2;
    glViewport(dx, 0, ClientSize.Height, ClientSize.Height);
} else {
    int dy = (ClientSize.Height - ClientSize.Width) / 2;
    glViewport(0, dy, ClientSize.Width, ClientSize.Width);
}
```

A.3.3.3. Setting a perspective projection of a scene

Self-study OpenGL actions and commands for displaying a scene in perspective projection.

A.3.3.4. Modeling object control and user interface

To organize the interface with the user, we use the keyboard and the mouse. Control of the coordinate system and scale will be connected with the events of the manipulator «mouse». Let's fix the pressing of the left button in the logical variable ***MoveAxes*** and save the current coordinates of the manipulator:

```
private void Mouse_Down(object sender, MouseEventArgs e)
{
    MoveAxes = (e.Button == MouseButtons.Left);
    dx = e.X; dy = e.Y;
}
```

Then, if the left button is not released and the manipulator is moving, we associate the horizontal movement with the rotation angle around the Y axis, and the vertical movement with the rotation angle around the X axis:

```
private void Mouse_Move(object sender, MouseEventArgs e)
{
    if (MoveAxes) {
        ay += (e.X - dx)/1.0;
        ax += (e.Y - dy)/1.0;
        dx = e.X; dy = e.Y;
        Invalidate();
    }
}
```

After that, we again save the current coordinates of the manipulator and inform the window about the need to redraw the work area using the ***Invalidate()*** method. Tracking the movements of the manipulator until the left button is released:

```
private void Mouse_Up(object sender, MouseEventArgs e)
{
    MoveAxes = MoveAxes && (e.Button != MouseButtons.Left);
}
```

In the same way, we track the necessary events of pressing the keys, linking the change in the angle of rotation ***az*** by one degree with the «Up» and «Down» keys, and the value of ***S*** – by the value ***ds*** ds with the keys «Left», «Right»:

```
private void Key_Down(object sender, PreviewKeyDownEventArgs e)
{
    if (e.KeyCode == Keys.Up)       az += 1;
    if (e.KeyCode == Keys.Down)     az -= 1;
    if (e.KeyCode == Keys.Left)     S -= ds;
    if (e.KeyCode == Keys.Right)    S += ds;
    Invalidate();
}
```

To control the zoom, we'll use the scroll wheel, using the ***e.Delta*** value as the zoom increment:

```
private void Mouse_Wheel(object sender, MouseEventArgs e)
{
    m += e.Delta / 2000.0;
    Invalidate();
}
```

A.3.3.5. Coordinate axes

To increase the overall clarity and simplify the orientation of the elements of the kinematic scheme, you can use the image of the coordinate axes as follows:

```
private void Axes(double s)
{
    glColor3d(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
        glVertex3d(0.0, 0.0, 0.0);  glVertex3d(s, 0.0, 0.0);
        glVertex3d(0.0, 0.0, 0.0);  glVertex3d(0.0, s, 0.0);
        glVertex3d(0.0, 0.0, 0.0);  glVertex3d(0.0, 0.0, s);
    glEnd();
    OutText("X", s, 0, 0);
    OutText("Y", 0, s, 0);
    OutText("Z", 0, 0, s);
}
```

A.3.3.6. Segment output

To display any segment in the initial state in accordance with (A.3.6) we use the following program code, adding ***r***, ***g***, ***b*** values to control the color of the rendered shape:

```
void Segment(double size, double r, double g, double b)
{
    glColor3d(r, g, b);  glLineWidth(5);
    glBegin(GL_LINES);
        glVertex3d(0.0, 0.0, 0.0);
        glVertex3d(0.0, size, 0.0);
    glEnd();
    glLineWidth(1);
}
```

Using quadric objects

The results of self-study that were used to solution this problem.

Defining light parameters

The results of self-study that were used to solution this problem.

Defining material parameters

The results of self-study that were used to solution this problem.

Surface texturing of scene objects

The results of self-study that were used to solution this problem.

Implementing shadow lighting from the manipulator model

The results of self-study that were used to solution this problem.

A.3.3.7. The order of affine transformations

In the final form, the image is formed in the method that is responsible for redrawing the entire work area:

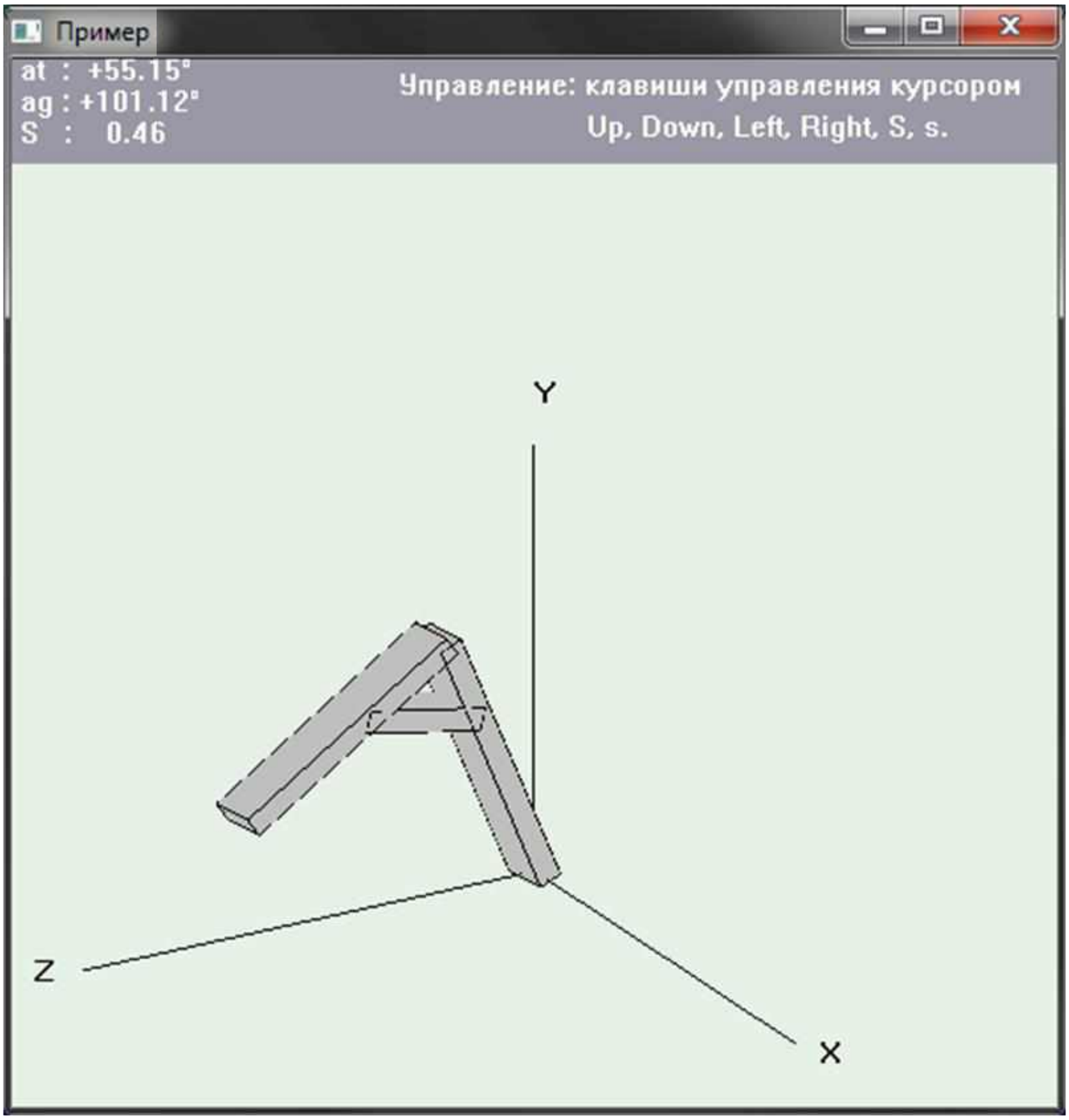
```
public override void OnRender()
{
    // Set the coordinate system
    glLoadIdentity();
    glOrtho(-2, 2, -2, 2, -2, 2);

     // Set general transformations Q
    glRotated(ax, 1, 0, 0);
    glRotated(ay, 0, 1, 0);
    glScaled(m, m, m);

    Axes(1.8); // // Render the coordinate axes

    glRotated(az, 0, 0, -1);
    Segment(a, 1,0,0);

    // Store the current transformation matrix on the stack
    glPushMatrix();
        glTranslated(0, a, 0);
        glRotated(ag, 0, 0, -1);
        Segment(b, 0, 0, 1);
    // Restore the current transformation matrix from the stack
    glPopMatrix();

    glTranslated(0, a2, 0);
    glRotated(at, 0, 0, -1);
    Segment(c, 0, 1, 0);

}
```

A.3.3.8. Project architecture

Brief information about the development environment of a software project. Description of the files included in the project and their purpose. A diagram of the developed classes (if the use of OOP is declared).

A.3.4. Results of the kinematic scheme model visualization

Examples of screenshot demonstrating the work of the program (examples on Fig. A.3.2 – A.3.4).


Пример

```
at :  +55.15°
ag : +101.12°
S  :    0.46
```

Управление: клавиши управления курсором
Up, Down, Left, Right, S, s.

Fig. A.3.2. Visualization of the task with minimal requirements

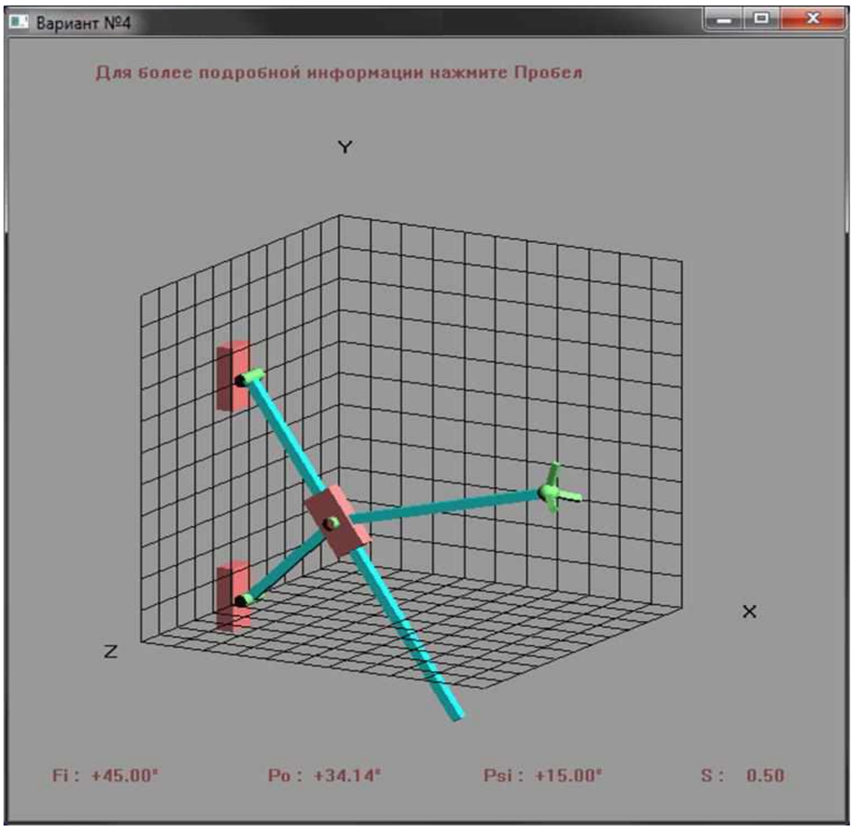


Fig. A.3.3. Example of rendering a task with lighting

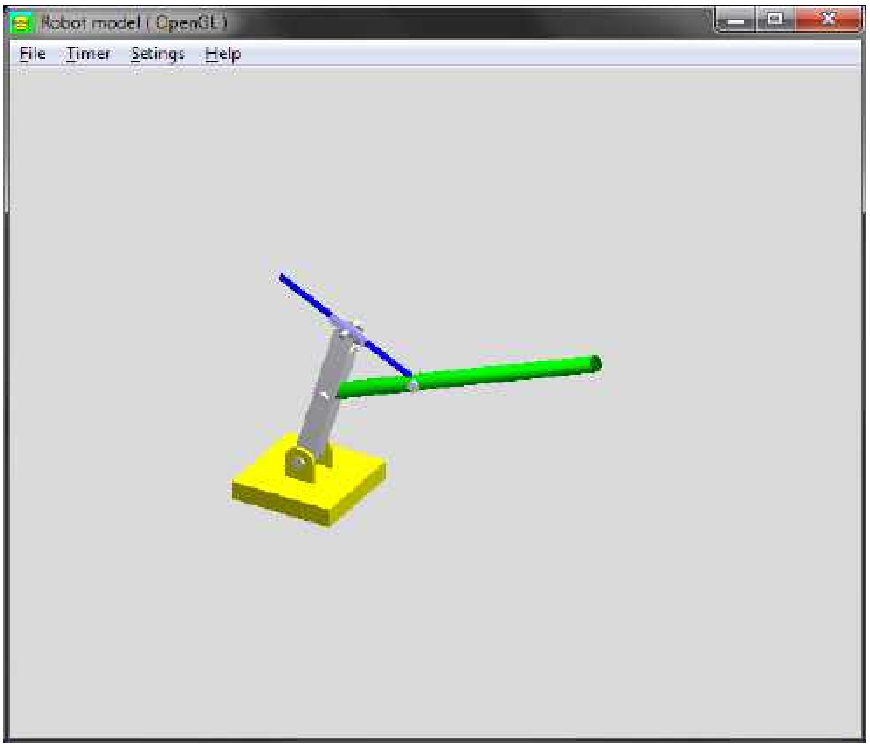


Fig. A.3.4. An example of visualizing a problem with quadric objects

A.3.5. Conclusions

A brief description of the software/tools are used and the achieved results.

BIBLIOGRAPHICS

Primary list

1. John Kessenich, Graham Sellers, Dave Shreiner. OpenGL Programming Guide, The Official Guide to Learning OpenGL, Version 4.5 9th edt. – Addison-Wesley Professional, 2016. – 976 p.
2. Hearn Baker Carithers, Computer Graphics with OpenGL, Fourth Edition. – Pearson Education Limited, 2014. – 812 p.
3. Mathematical Elements for Computer Graphics (2nd Edition) / David F. Rogers, J. Alan Adams; – McGraw-Hill Science/Engineering/Math, 1989. – 611 p.
4. Performing Selection and Feedback [electronic resource] // – Access: https://learn.microsoft.com/en-us/windows/win32/opengl/performing-selection-and-feedback.
5. Procedural elements for computer graphics by David F. Rogers, 1998, WCB/McGraw-Hill edition, in English - 2nd ed. 1998. – 711 p.
6. Francis S. Hill, Computer Graphics: Using OpenGL. – Prentice Hall, 2001. – 922 p.

Additional list

7. Discontinuities [Electronic resource] // WolframAlpha Computational intelligence. – Access : https://www.wolframalpha.com/input/?i=discontinuities+1%2F(x-1).
8. Mark de Berg,·Computational Geometry. Algorithms and Applications. Third Edition / Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. – Springer, 2008. – 386 p.
9. Windows Graphics Programming: Win32 GDI and DirectDraw / Feng Yuan. – Hewlett-Packard Professional Books, 2000. – 1234 p.
10. Jeffrey Richter, Programming Applications for Microsoft Windows. – Microsoft Press, 1999. – 1056 p.
11. Robert Nystrom, Game Programming Patterns. – Genever Benning, November 2, 2014. – 354 p.
12. How is the .NET event, Application.OnIdle, coded in Win32? [electronic resource] // StackOverflow. – Access : https://stackoverflow.com/questions/3331012/how-is-the-net-event- application-onidle-coded-in-win32.

CONTENT

Навчальне видання

Лучшев Павло Олександрович
Голуб Надія Григорівна

КОМП'ЮТЕРНА ГРАФІКА

(Англійською мовою)

Редактор В. В. Рижкова
Технічний редактор А. М. Ємленінова

Зв. план, 2024
Підписано до видання 28.11.2024
Ум. друк. арк. 4,4. Обл.-вид. арк. 5. Електронний ресурс

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001