Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №4 по теме: «Аффинные преобразования в пространстве»

**Выполнил:**  
студент группы ПВ-41  
Адаменко И. И.

**Проверил:**  
старший преподаватель  
Осипов О. В.

Белгород  
2015

**Цель работы:** получение навыков построения аффинных преобразований в пространстве и разработка приложения с использование GDI.

# Задание

Разработать алгоритм и составить программу для построения на экране проекций выбранной фигуры (не менее 15 граней). Рабочая область программы должна быть поделена на 4 части:

* вид спереди;
* вид сбоку;
* вид сверху;
* объёмный вид, который можно вращать с помощью мыши или клавиатуры.

В программе должны выводиться следующие виды проекций (в объёмном виде):

* параллельная;
* косоугольная (свободная и кабинетная);
* центральная.

Также в программе должны быть реализованы модули для работы с матрицами.

# Теоретическая часть

Аффинные преобразования в трёхмерном пространстве осуществляются тем же образом, что и в двумерном, однако, вместо матриц 3×3 используются матрицы размерностью 4×4. Данный переход к новому способу задания точек даёт возможность воспользоваться матричной записью и в более сложных трехмерных задачах. Любое аффинное преобразование в трёхмерном пространстве может быть представлено в виде комбинации вращений, растяжений, отражений и переносов.

Матрица вращений вокруг оси абсцисс на угол :

Матрица вращения вокруг оси ординат на угол :

Матрица вращения вокруг оси аппликат на угол :

Матрица сжатия/растяжения (масштабирование):

— коэффициент сжатия относительно оси X (остальные аналогично Y и Z).

Матрица отражения относительно плоскости XY:

Для остальных плоскостей она строится аналогичным образом, меняется только положение −1.

Матрица переноса:

— переносы, соответственно, по осям X, Y, Z.

Изображение объектов на экране связано с ещё одной геометрической операцией — проектированием при помощи пучка прямых. В компьютерной графике используется несколько различных видов проектирования. Чаще всего применяются параллельное и центральное.

Параллельные:

1. Ортографическая — пучок проецирует объект в реальных размерах.
2. Аксонометрическая (триметрическая, диметрическая и изометрическая) — как ортографическая, но объект располагается под разными углами.
3. Косоугольная — пучок падает на плоскость под углом.

a. Свободная (проекция Кавалье)

b. Кабинетная (проекция Кабине)

Перспективные:

1. Одноточечная
2. Двухточечная
3. Трёхточечная (центральная) — пучок выходит из одной точки. При бесконечном удалении точки от объекта вырождается в ортографическую.

Все проекции можно получить с помощью аффинных преобразований, умножением на матрицу проекции.

## Преобразования

В этой лабораторной работе использовались следующие матричные преобразования:

1. Созданные направляющие (четыре прямые, в местах пересечения которых находятся центры четырёх частей рабочей области программы) переносились на то место, в котором они должны быть отрисованы.

Таким образом, для каждой созданной прямой совершались следующие трансформации:

1. Вид спереди для фигуры подвергался следующим трансформациям:
   1. Масштабировался до необходимого размера (размер высчитывается в зависимости от размера рабочей области программы).
   2. Переносился таким образом, чтобы его центр совпадал с центром необходимой ¼ рабочей области.

Таким образом, для вида спереди совершались следующие трансформации:

1. Вид сбоку для фигуры подвергался следующим трансформациям:
   1. Масштабировался до необходимого размера (размер высчитывается в зависимости от размера рабочей области программы).
   2. Переносился так, чтобы его центр был в начале координат.
   3. Поворачивался вокруг оси ординат на 90 градусов.
   4. Переносился обратно.
   5. Переносился таким образом, чтобы его центр совпадал с центром необходимой ¼ рабочей области.

Таким образом, для вида сбоку совершались следующие трансформации:

1. Вид снизу для фигуры подвергался следующим трансформациям:
   1. Масштабировался до необходимого размера (размер высчитывается в зависимости от размера рабочей области программы).
   2. Переносился так, чтобы его центр был в начале координат.
   3. Поворачивался вокруг оси ординат на 90 градусов.
   4. Поворачивался вокруг оси абсцисс на 90 градусов.
   5. Переносился обратно.
   6. Переносился таким образом, чтобы его центр совпадал с центром необходимой ¼ рабочей области.

Таким образом, для вида снизу совершались следующие трансформации:

1. Трёхмерная модель фигуры подвергалась следующим трансформациям:
   1. Масштабировалась до необходимого размера (размер высчитывается в зависимости от размера рабочей области программы).
   2. Переносилась так, чтобы его центр был в начале координат.
   3. Поворачивалась вокруг оси абсцисс на X градусов.
   4. Поворачивалась вокруг оси ординат на Y градусов.
   5. Переносилась обратно.
   6. Трансформировалась с помощью матрицы соответствующей проекции.
   7. Переносилась таким образом, чтобы её центр совпадал с центром необходимой ¼ рабочей области.

X и Y — количество градусов, которое указал пользователь (с помощью клавиш-стрелок на клавиатуре)

Таким образом, для трёхмерной модели совершались следующие трансформации:

# Практическая часть

## Код программы

Main.cs — код основной программы:

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Drawing;
4. **using** System.Windows.Forms;
6. **namespace** lab4
7. {
8. **public** partial **class** Main : Form
9. {
10. **private** Graphics g;
11. **private** P leftTop;
12. **private** P rightTop;
13. **private** P leftBottom;
14. **private** P rightBottom;
15. **private** **float** width;
16. **private** **float** height;
17. **private** **float** scale;
18. **private** **float** rotateX = 0;
19. **private** **float** rotateY = 0;
20. **private** **enum** Types
21. {
22. Orthographic,
23. ObliqueCavalier,
24. ObliqueCabinet,
25. Perspective
26. }
27. **private** Types currentType = Types.Orthographic;
28. **private** S shape;
29. **private** P shapeCenter = **null**;
30. **private** **float** speed = 3F;
31. **private** **float** cmInForm = 90F;
33. **public** Main()
34. {
35. InitializeComponent();
36. **this**.SetStyle(
37. ControlStyles.DoubleBuffer |
38. ControlStyles.UserPaint |
39. ControlStyles.AllPaintingInWmPaint |
40. ControlStyles.ResizeRedraw,
41. **true**
42. );
43. **this**.UpdateStyles();
44. }
46. **private** **void** Canvas\_Paint(**object** sender, PaintEventArgs e)
47. {
48. g = e.Graphics;
49. width = g.ClipBounds.Width;
50. height = g.ClipBounds.Height;
52. leftTop = **new** P(width / 4, height / 4, 0);
53. rightTop = **new** P(3 \* width / 4, height / 4, 0);
54. leftBottom = **new** P(width / 4, 3 \* height / 4, 0);
55. rightBottom = **new** P(3 \* width / 4, 3 \* height / 4, 0);
57. DrawAxis();
59. CalcScale(width, height);
61. shape = CreateFigure();
62. shape \*= **new** MScale(scale, scale, scale);
64. **if** (shapeCenter == **null**)
65. {
66. shapeCenter = shape.Center;
67. }
69. var shapeSide = shape.Clone();
70. var shapeFront = shape.Clone();
71. var shapeBottom = shape.Clone();
73. shapeFront \*= **new** MTranslate(leftTop.X - shapeFront.Center.X,
74. leftTop.Y - shapeFront.Center.Y,
75. leftTop.Z - shapeFront.Center.Z);
76. shapeFront.Draw(g);
78. shapeSide \*= **new** MTranslate(-shapeSide.Center.X,
79. -shapeSide.Center.Y,
80. -shapeSide.Center.Z);
81. shapeSide \*= **new** MRotate(90, 'y');
82. shapeSide \*= **new** MTranslate(shapeSide.Center.X,
83. shapeSide.Center.Y,
84. shapeSide.Center.Z);
85. shapeSide \*= **new** MTranslate(rightTop.X - shapeSide.Center.X,
86. rightTop.Y - shapeSide.Center.Y,
87. rightTop.Z - shapeSide.Center.Z);
88. shapeSide.Draw(g);
90. shapeBottom \*= **new** MTranslate(-shapeBottom.Center.X,
91. -shapeBottom.Center.Y,
92. -shapeBottom.Center.Z);
93. shapeBottom \*= **new** MRotate(-90, 'y');
94. shapeBottom \*= **new** MRotate(-90, 'x');
95. shapeBottom \*= **new** MTranslate(shapeBottom.Center.X,
96. shapeBottom.Center.Y,
97. shapeBottom.Center.Z);
98. shapeBottom \*= **new** MTranslate(leftBottom.X - shapeBottom.Center.X,
99. leftBottom.Y - shapeBottom.Center.Y,
100. leftBottom.Z - shapeBottom.Center.Z);
101. shapeBottom.Draw(g);
103. shape \*= **new** MTranslate(-shapeCenter.X, -shapeCenter.Y, -shapeCenter.Z);
104. shape \*= **new** MRotate(rotateX, 'x');
105. shape \*= **new** MRotate(rotateY, 'y');
106. shape \*= **new** MTranslate(shapeCenter.X, shapeCenter.Y, shapeCenter.Z);
108. **switch** (currentType)
109. {
110. **case** Types.Orthographic:
111. shape \*= **new** MOrthographicProjection("xy");
112. **break**;
113. **case** Types.ObliqueCavalier:
114. shape \*= **new** MObliqueСavalierProjection();
115. **break**;
116. **case** Types.ObliqueCabinet:
117. shape \*= **new** MObliqueCabinetProjection();
118. **break**;
119. **case** Types.Perspective:
120. shape \*= **new** MPerspectiveProjection(2000, 2000, 2000);
121. **break**;
122. }
124. shape \*= **new** MTranslate(rightBottom.X - shapeCenter.X,
125. rightBottom.Y - shapeCenter.Y,
126. rightBottom.Z - shapeCenter.Z);
128. shape.Draw(g);
129. }
131. **private** S CreateFigure()
132. {
133. var s = **new** S(**new** List<P>
134. {
135. // 0
136. **new** P(0, 0, 0),
137. // 1
138. **new** P(2, 0, 0),
139. // 2
140. **new** P(5, 4, 0),
141. // 3
142. **new** P(8, 4, 0),
143. // 4
144. **new** P(10, 6, 0),
145. // 5
146. **new** P(10, 9, 0),
147. // 6
148. **new** P(5, 9, 0),
149. // 7
150. **new** P(5, 9, 1),
151. // 8
152. **new** P(9, 9, 1),
153. // 9
154. **new** P(5, 13, 1),
155. // 10
156. **new** P(9, 13, 1),
157. // 11
158. **new** P(5, 13, 0),
159. // 12
160. **new** P(9, 13, 3),
161. // 13
162. **new** P(5, 13, 3),
163. // 14
164. **new** P(5, 9, 3),
165. // 15
166. **new** P(9, 9, 3),
167. // 16
168. **new** P(10, 9, 4),
169. // 17
170. **new** P(5, 9, 4),
171. // 18
172. **new** P(5, 13, 4),
173. // 19
174. **new** P(0, 13, 4),
175. // 20
176. **new** P(0, 13, 0),
177. // 21
178. **new** P(0, 0, 4),
179. // 22
180. **new** P(2, 0, 4),
181. // 23
182. **new** P(5, 4, 4),
183. // 24
184. **new** P(8, 4, 4),
185. // 25
186. **new** P(10, 6, 4)
187. });
189. s.Pairs = **new** List<Tuple<**int**, **int**>>
190. {
191. **new** Tuple<**int**, **int**>(0, 1),
192. **new** Tuple<**int**, **int**>(0, 21),
193. **new** Tuple<**int**, **int**>(0, 20),
194. **new** Tuple<**int**, **int**>(1, 2),
195. **new** Tuple<**int**, **int**>(1, 22),
196. **new** Tuple<**int**, **int**>(2, 3),
197. **new** Tuple<**int**, **int**>(2, 23),
198. **new** Tuple<**int**, **int**>(3, 4),
199. **new** Tuple<**int**, **int**>(3, 24),
200. **new** Tuple<**int**, **int**>(4, 5),
201. **new** Tuple<**int**, **int**>(4, 25),
202. **new** Tuple<**int**, **int**>(5, 6),
203. **new** Tuple<**int**, **int**>(5, 16),
204. **new** Tuple<**int**, **int**>(6, 7),
205. **new** Tuple<**int**, **int**>(6, 11),
206. **new** Tuple<**int**, **int**>(7, 8),
207. **new** Tuple<**int**, **int**>(7, 9),
208. **new** Tuple<**int**, **int**>(8, 10),
209. **new** Tuple<**int**, **int**>(8, 15),
210. **new** Tuple<**int**, **int**>(9, 11),
211. **new** Tuple<**int**, **int**>(9, 10),
212. **new** Tuple<**int**, **int**>(10, 12),
213. **new** Tuple<**int**, **int**>(11, 20),
214. **new** Tuple<**int**, **int**>(12, 13),
215. **new** Tuple<**int**, **int**>(12, 15),
216. **new** Tuple<**int**, **int**>(13, 14),
217. **new** Tuple<**int**, **int**>(13, 18),
218. **new** Tuple<**int**, **int**>(14, 15),
219. **new** Tuple<**int**, **int**>(14, 17),
220. **new** Tuple<**int**, **int**>(16, 17),
221. **new** Tuple<**int**, **int**>(16, 25),
222. **new** Tuple<**int**, **int**>(17, 18),
223. **new** Tuple<**int**, **int**>(18, 19),
224. **new** Tuple<**int**, **int**>(19, 20),
225. **new** Tuple<**int**, **int**>(19, 21),
226. **new** Tuple<**int**, **int**>(21, 22),
227. **new** Tuple<**int**, **int**>(22, 23),
228. **new** Tuple<**int**, **int**>(23, 24),
229. **new** Tuple<**int**, **int**>(24, 25),
230. };
232. **return** s;
233. }
235. **private** **void** DrawAxis()
236. {
237. var line = **new** S(**new** List<P>()
238. {
239. **new** P(0, height / 4, 0),
240. **new** P(width, height / 4, 0)
241. });
243. line.Pen = **new** Pen(Color.LightGray, 1F);
244. line.Draw(g);
246. line \*= **new** MTranslate(0, height / 2, 0);
247. line.Draw(g);
249. line = **new** S(**new** List<P>()
250. {
251. **new** P(width / 4, 0, 0),
252. **new** P(width / 4, height, 0)
253. });
255. line.Pen = **new** Pen(Color.LightGray, 1F);
256. line.Draw(g);
258. line \*= **new** MTranslate(width / 2, 0, 0);
259. line.Draw(g);
260. }
262. **private** **void** CalcScale(**float** width, **float** height)
263. {
264. var min = width < height ? width : height;
266. scale = min / (cmInForm / 2);
267. }
269. **protected** **override** **void** OnKeyDown(KeyEventArgs e)
270. {
271. **switch** (e.KeyCode)
272. {
273. **case** Keys.Left:
274. rotateY -= speed;
275. shapeCenter = **null**;
276. Canvas.Refresh();
277. **break**;
278. **case** Keys.Right:
279. rotateY += speed;
280. shapeCenter = **null**;
281. Canvas.Refresh();
282. **break**;
283. **case** Keys.Up:
284. rotateX += speed;
285. shapeCenter = **null**;
286. Canvas.Refresh();
287. **break**;
288. **case** Keys.Down:
289. rotateX -= speed;
290. shapeCenter = **null**;
291. Canvas.Refresh();
292. **break**;
293. **case** Keys.D1:
294. currentType = Types.Orthographic;
295. **this**.Text = "1. Ортографическая проекция";
296. shapeCenter = **null**;
297. Canvas.Refresh();
298. **break**;
299. **case** Keys.D2:
300. currentType = Types.ObliqueCavalier;
301. **this**.Text = "2. Косоугольная свободная проекция";
302. shapeCenter = **null**;
303. Canvas.Refresh();
304. **break**;
305. **case** Keys.D3:
306. currentType = Types.ObliqueCabinet;
307. **this**.Text = "3. Косоугольная кабинетная проекция";
308. shapeCenter = **null**;
309. Canvas.Refresh();
310. **break**;
311. **case** Keys.D4:
312. currentType = Types.Perspective;
313. **this**.Text = "4. Перспективная проекция";
314. shapeCenter = **null**;
315. Canvas.Refresh();
316. **break**;
317. }
318. }
320. **private** **void** Main\_Resize(**object** sender, EventArgs e)
321. {
322. shapeCenter = **null**;
323. }
324. }
325. }

M.cs — базовый класс, реализующий работу с матрицами:

1. **using** System.Collections.Generic;
2. **using** System.Linq;
4. **namespace** lab4
5. {
6. **class** M
7. {
8. **protected** **float**[,] Matrix { **get**; **set**; }
10. **public** M()
11. {
12. Matrix = **new** **float**[,] {
13. {0, 0, 0, 0},
14. {0, 0, 0, 0},
15. {0, 0, 0, 0},
16. {0, 0, 0, 0},
17. };
18. }
20. **public** M(**float**[,] matrix)
21. {
22. Matrix = matrix;
23. }
25. **public** **float**[] VectorMult(**float**[] vector)
26. {
27. var result = **new** List<**float**>();
29. **for** (**int** i = 0; i < Matrix.GetLength(0); i++)
30. {
31. var res = 0F;
33. **for** (**int** j = 0; j < Matrix.GetLength(1); j++)
34. {
35. res += vector[j] \* Matrix[j, i];
36. }
38. result.Add(res);
39. }
41. **return** result.Select(x => x / result[result.Count - 1]).ToArray();
42. }
43. }
44. }

MObliqueProjection.cs — базовый класс (дочерний для M), реализующий работу матриц косоугольного проецирования:

1. **namespace** lab4
2. {
3. **class** MObliqueProjection : M
4. {
5. **public** MObliqueProjection(**float** a)
6. : **base**(**new** **float**[,] {
7. { 1, 0, 0, 0 },
8. { 0, 1, 0, 0 },
9. { a, a, 1, 0 },
10. { 0, 0, 0, 1 },
11. })
12. {
13. // nothing here :)
14. }
15. }
16. }

MObliqueСavalierProjection.cs — дочерний для MObliqueProjection класс, реализующий работу матриц косоугольного свободного проецирования:

1. **using** System;
3. **namespace** lab4
4. {
5. **class** MObliqueСavalierProjection : MObliqueProjection
6. {
7. **public** MObliqueСavalierProjection()
8. : **base**((**float**)Math.Cos(Math.PI / 4))
9. {
10. // nothing here :)
11. }
12. }
13. }

MObliqueCabinetProjection.cs — дочерний для MObliqueProjection, реализующий работу матриц косоугольного кабинетного проецирования:

1. **using** System;
3. **namespace** lab4
4. {
5. **class** MObliqueCabinetProjection : MObliqueProjection
6. {
7. **public** MObliqueCabinetProjection()
8. : **base**((**float**)Math.Cos(Math.PI / 4) / 2)
9. {
10. // nothing here :)
11. }
12. }
13. }

MOrthographicProjection.cs — дочерний для M класс, реализующий работу матриц ортографического проецирования:

1. **namespace** lab4
2. {
3. **class** MOrthographicProjection : M
4. {
5. **public** MOrthographicProjection(**string** type)
6. {
7. **float** x, y, z;
8. x = y = z = 1;
10. **switch** (type)
11. {
12. **case** "xy": z = 0;
13. **break**;
14. **case** "yz": x = 0;
15. **break**;
16. **case** "xz": y = 0;
17. **break**;
18. }
20. **this**.Matrix = **new** **float**[,] {
21. { x, 0, 0, 0 },
22. { 0, y, 0, 0 },
23. { 0, 0, z, 0 },
24. { 0, 0, 0, 1 },
25. };
26. }
27. }
28. }

MPerspectiveProjection.cs — дочерний для M класс, реализующий работу матриц перспективного центрального проецирования:

1. **namespace** lab4
2. {
3. **class** MPerspectiveProjection : M
4. {
5. **public** MPerspectiveProjection(**float** a, **float** b, **float** c)
6. {
7. **this**.Matrix = **new** **float**[,] {
8. { 1, 0, 0, -1 / a },
9. { 0, 1, 0, -1 / b },
10. { 0, 0, 1, -1 / c },
11. { 0, 0, 0, 1 },
12. };
13. }
14. }
15. }

MReflect.cs — дочерний для M класс, реализующий работу матриц для операции отражения:

1. **namespace** lab4
2. {
3. **class** MReflect : M
4. {
5. **public** MReflect(**string** type)
6. {
7. **float** x, y, z;
8. x = y = z = 1;
10. **switch** (type)
11. {
12. **case** "xy": z = -1;
13. **break**;
14. **case** "yz": x = -1;
15. **break**;
16. **case** "xz": y = -1;
17. **break**;
18. }
20. **this**.Matrix = **new** **float**[,] {
21. { x, 0, 0, 0 },
22. { 0, y, 0, 0 },
23. { 0, 0, z, 0 },
24. { 0, 0, 0, 1 },
25. };
26. }
27. }
28. }

MRotate.cs — дочерний для M класс, реализующий работу матриц для операции поворота:

1. **using** System;
3. **namespace** lab4
4. {
5. **class** MRotate : M
6. {
7. **public** MRotate(**double** deg, **char** type)
8. {
9. var phi = Math.PI \* deg / 180;
10. var cos = (**float**)Math.Cos(phi);
11. var sin = (**float**)Math.Sin(phi);
13. **switch** (type)
14. {
15. **case** 'x':
16. **this**.Matrix = **new** **float**[,] {
17. { 1, 0, 0, 0 },
18. { 0, cos, sin, 0 },
19. { 0, -sin, cos, 0 },
20. { 0, 0, 0, 1 },
21. };
22. **break**;
23. **case** 'y':
24. **this**.Matrix = **new** **float**[,] {
25. { cos, 0, -sin, 0 },
26. { 0, 1, 0, 0 },
27. { sin, 0, cos, 0 },
28. { 0, 0, 0, 1 },
29. };
30. **break**;
31. **case** 'z':
32. **this**.Matrix = **new** **float**[,] {
33. { cos, sin, 0, 0 },
34. { -sin, cos, 0, 0 },
35. { 0, 0, 1, 0 },
36. { 0, 0, 0, 1 },
37. };
38. **break**;
39. }
40. }
41. }
42. }

MScale.cs — дочерний для M класс, реализующий работу матриц для операции масштабирования:

1. **namespace** lab4
2. {
3. **class** MScale : M
4. {
5. **public** MScale(**float** a, **float** b, **float** c)
6. : **base**(**new** **float**[,] {
7. { a, 0, 0, 0 },
8. { 0, b, 0, 0 },
9. { 0, 0, c, 0 },
10. { 0, 0, 0, 1 },
11. })
12. {
13. // nothing here :)
14. }
15. }
16. }

MTranslate.cs — дочерний для M класс, реализующий работу матриц для операции переноса:

1. **namespace** lab4
2. {
3. **class** MTranslate : M
4. {
5. **public** MTranslate(**float** x, **float** y, **float** z)
6. : **base**(**new** **float**[,] {
7. { 1, 0, 0, 0 },
8. { 0, 1, 0, 0 },
9. { 0, 0, 1, 0 },
10. { x, y, z, 1 }
11. })
12. {
13. // nothing here :)
14. }
15. }
16. }

P.cs — класс, реализующий работу с точками фигур:

1. **using** System.Drawing;
3. **namespace** lab4
4. {
5. **class** P
6. {
7. **public** **float** X { **get**; **set**; }
8. **public** **float** Y { **get**; **set**; }
9. **public** **float** Z { **get**; **set**; }
11. **public** P(**float** x, **float** y, **float** z)
12. {
13. X = x;
14. Y = y;
15. Z = z;
16. }
18. **public** P Clone()
19. {
20. **return** **new** P(**this**.X, **this**.Y, **this**.Z);
21. }
23. **public** PointF ToPointF()
24. {
25. **return** **new** PointF(**this**.X, **this**.Y);
26. }
27. }
28. }

S.cs — класс, реализующий работу с фигурами:

1. **using** System.Collections.Generic;
2. **using** System.Drawing;
3. **using** System;
5. **namespace** lab4
6. {
7. **class** S
8. {
9. **public** List<P> Points { **get**; **set**; }
10. **public** List<Tuple<**int**, **int**>> Pairs { **get**; **set**; }
11. **public** P Center
12. {
13. **get**
14. {
15. var min = Points[0].Clone();
16. var max = Points[0].Clone();
18. **for** (**int** i = 0; i < Points.Count; i++)
19. {
20. **if** (Points[i].X < min.X) min.X = Points[i].X;
21. **if** (Points[i].Y < min.Y) min.Y = Points[i].Y;
22. **if** (Points[i].Z < min.Z) min.Z = Points[i].Z;
23. **if** (Points[i].X > max.X) max.X = Points[i].X;
24. **if** (Points[i].Y > max.Y) max.Y = Points[i].Y;
25. **if** (Points[i].Z > max.Z) max.Z = Points[i].Z;
26. }
28. **return** **new** P((max.X + min.X) / 2,
29. (max.Y + min.Y) / 2,
30. (max.Z + min.Z) / 2);
31. }
32. }
33. **public** Pen Pen { **get**; **set**; }
35. **public** S()
36. {
37. Pen = **new** Pen(Color.Black, 2F);
38. Points = **new** List<P>();
39. Pairs = **null**;
40. }
42. **public** S(List<P> points)
43. {
44. Pen = **new** Pen(Color.Black, 2F);
45. Points = points;
46. Pairs = **null**;
47. }
49. **public** **void** Draw(Graphics g)
50. {
51. **if** (Pairs != **null**)
52. {
53. **foreach** (var pair **in** Pairs)
54. {
55. g.DrawLine(Pen,
56. Points[pair.Item1].ToPointF(),
57. Points[pair.Item2].ToPointF());
58. }
60. **return**;
61. }
63. **for** (**int** i = 1; i < Points.Count; i++)
64. {
65. g.DrawLine(Pen,
66. Points[i - 1].ToPointF(),
67. Points[i].ToPointF());
68. }
69. }
71. **public** S Clone()
72. {
73. var clone = **new** S();
75. **for** (**int** i = 0; i < **this**.Points.Count; i++)
76. {
77. clone.Points.Add(Points[i].Clone());
78. }
80. **if** (Pairs != **null**)
81. {
82. clone.Pairs = **new** List<Tuple<**int**, **int**>>();
84. **foreach** (var pair **in** Pairs)
85. {
86. clone.Pairs.Add(**new** Tuple<**int**, **int**>(pair.Item1, pair.Item2));
87. }
88. }
90. clone.Pen = **new** Pen(**this**.Pen.Color, **this**.Pen.Width);
92. **return** clone;
93. }
95. **public** **static** S **operator** \*(S s, M m)
96. {
97. var newS = s.Clone();
99. **for** (**int** i = 0; i < s.Points.Count; i++)
100. {
101. var point = **new** **float**[] { s.Points[i].X, s.Points[i].Y, s.Points[i].Z, 1 };
103. point = m.VectorMult(point);
105. newS.Points[i] = **new** P(point[0], point[1], point[2]);
106. }
108. **return** newS;
109. }
110. }
111. }

## Скриншоты

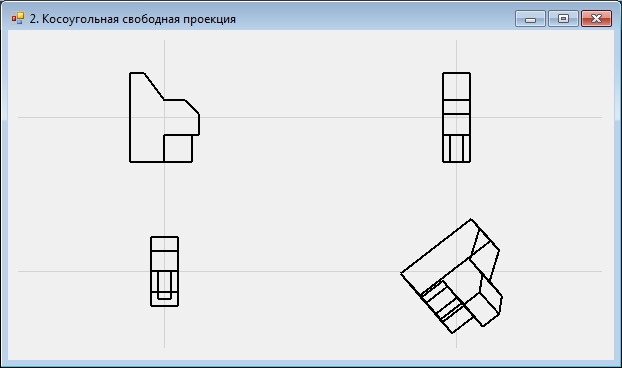


Рисунок 1. Рабочая программа