Министерство науки и высшего образования Российской Федерации
Лысьвенский филиал федерального государственного бюджетного образовательного учреждения высшего образования
«Пермский национальный исследовательский политехнический университет»

Кафедра технических дисциплин (ТД)

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

«Компьютерная графика»

основной профессиональной образовательной программы
подготовки бакалавров
по направлению 09.03.01
Информатика и вычислительная техника

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации лабораторных работ

Лысьва 2021 г.

Составители: ст.преподаватель кафедры ТД М.Н. Апталаев, Е.Л. Федосеева

Методические указания рассмотрены и одобрены на заседании кафедры ТД «30» августа 2021 г., протокол № 01.

СОДЕРЖАНИЕ

1 ПЕРЕЧЕНЬ ТЕМ ЛАБОРАТОРНЫХ РАБОТ 4
2 РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ И ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ 5
Лабораторная работа 1 Исследование аффинных преобразований 5
Лабораторная работа 2 Преобразование координат на плоскости 13
Лабораторная работа 3 Трехмерные преобразования 35
Лабораторная работа 4 Растровые алгоритмы построения контура фигур 63
Лабораторная работа 5 Алгоритмы отсечения фигур 74
Лабораторная работа 6 Растровые алгоритмы заполнения фигур 84
Лабораторная работа 7 Построение поверхностей 90
Лабораторная работа 8 Реализация алгоритмов растровой развертки линий 95
Лабораторная работа 9 Реализация преобразований по отношению к плоской фигуре 104
Лабораторная работа 10 Реализация преобразований по отношению к объемной фигуре 107
Лабораторная работа 11 Возможности интерфейса программирования, предоставляемого стандартом OpenGL 109
Приложение А 168

1 ПЕРЕЧЕНЬ ТЕМ ЛАБОРАТОРНЫХ РАБОТ

Таблица 1– Темы лабораторных работ

№ п.п.	Номер темы дисциплины	Наименование темы лабораторной работы
1.	2	Исследование аффинных преобразований
2.	3	Преобразование координат на плоскости
3.	4	Трехмерные преобразования
4.	5	Растровые алгоритмы построения контура фигур
5.	6	Алгоритмы отсечения фигур
6.	7	Растровые алгоритмы заполнения фигур
7.	8	Построение поверхностей
8.	9	Реализация алгоритмов растровой развертки линий
9.	10	Реализация преобразований по отношению к плоской фигуре
10.	11	Реализация преобразований по отношению к объемной фигуре
11.	12	Возможности интерфейса программирования, предоставляемого стандартом OpenGL

2 РЕКОМЕНДАЦИИ ПО ПОДГОТОВКЕ И ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1 Исследование аффинных преобразований

Цель работы: освоить математические методы исследования аффинных преобразований

Теоретическая часть

Для описания зрительных систем высших животных и человека и большинства систем компьютерного зрения могут быть применены схемы, представленные на рисунке 1, соответственно. В них точки поверхностей объектов реального мира проецируются с помощью сферической линзы на светочувствительные элементы (палочки и колбочки на внутренней поверхности глаза или элементы светочувствительной матрицы). По законам геометрической оптики такое проецирование осуществляется через точку, соответствующую центру линзы, т.е. проецирование производится непараллельными лучами. В целях упрощения анализа считается, что проецируются точки плоской поверхности объекта (что для большинства объектов реального мира выполняется по крайней мере локально) на плоскую матрицу светочувствительных элементов (что обычно соответствует истине, если мы имеем дело с системами компьютерного зрения, и справедливо для локальных участков сетчатки глаза).

1.1. Если проецируемая поверхность объекта параллельна плоскости, содержащей светочувствительные элементы, как это показано на рисунке 1 (а, б), сформированное изображение объекта, подлежащее дальнейшей обработке и распознаванию, геометрически трансформируется преобразованием подобия, включающим в себя линейные сдвиги вдоль декартовых координатных осей, равномерное масштабирование, вращение в плоскости изображения и зеркальное отражение относительно произвольной прямой, лежащей в плоскости изображения и проходящей в ней через начало координат. Масштабирование, вращение и линейные смещения проекции объекта на светочувствительной среде возникают при соответствующих пространственных перемещениях объекта в поле зрения системы относительно его эталонного пространственного положения. Зеркальное отражение, если не рассматривать тривиальное отражение через зеркало, может возникать при наблюдении прозрачного объекта с обратной (задней) стороны, что в реальных ситуациях встречается нечасто.

1.2. Если плоскость, содержащая светочувствительные элементы, становится не параллельна отображаемой поверхности объекта (поворачивается относительно некоторой оси, лежащей в этой плоскости), но объект

находится далеко от объектива (линзы), к указанным выше составляющим преобразования подобия добавляется сжатие изображения вдоль прямой, перпендикулярной указанной оси. Полученное более сложное геометрическое преобразование называется аффинным.

Аффинное геометрическое преобразование описывается при этом следующим выражением:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \Leftrightarrow \begin{cases} x' = a_1 x + a_2 y + a_3 \\ y' = a_4 x + a_5 y + a_6 \end{cases},$$

где x, y – абсцисса и ордината точки исходного изображения, x', y' – преобразованные координаты этой точки, $a_1 \ldots a_6$ – параметры преобразования.

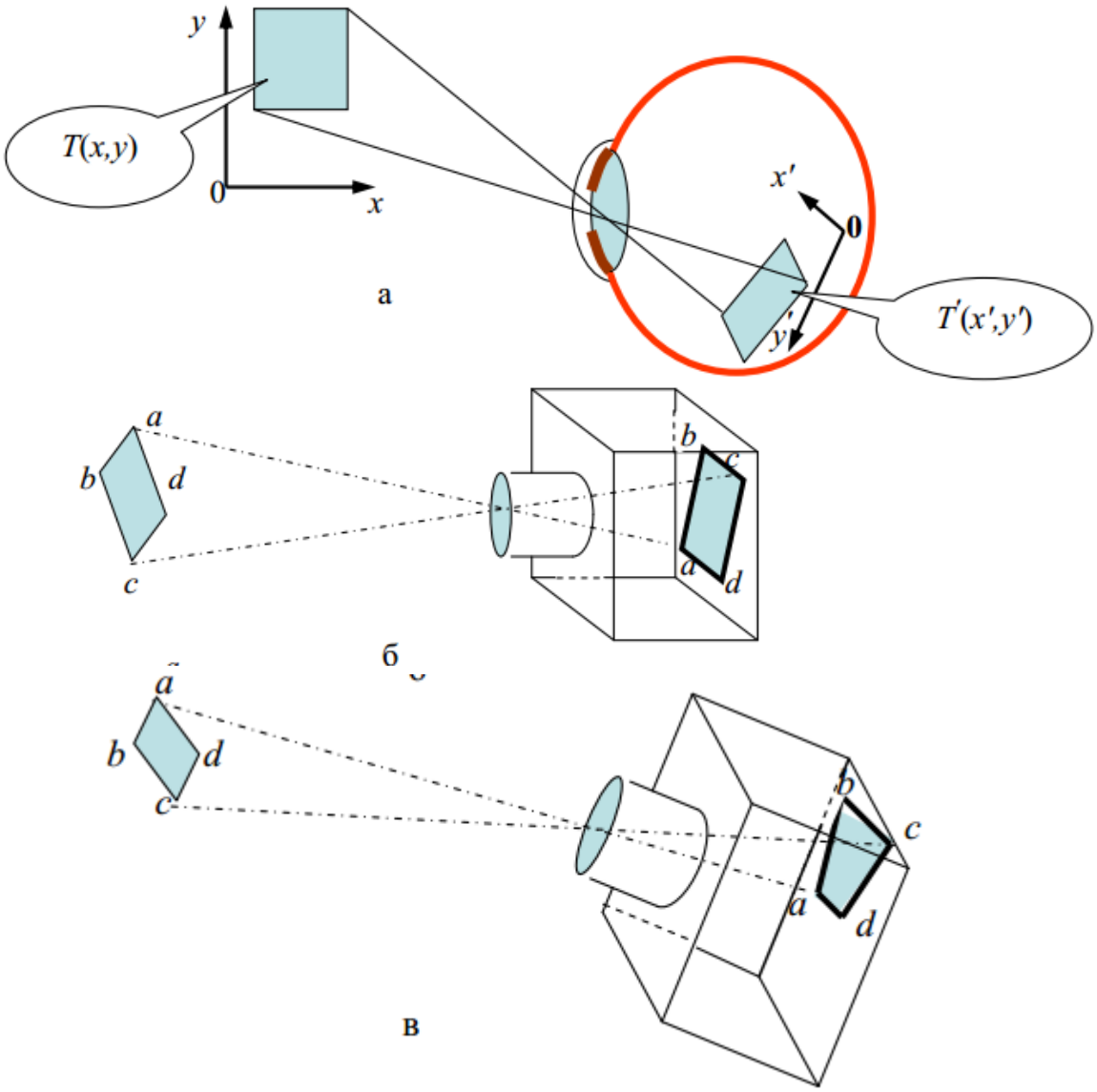


Рисунок 1 – Проецирование изображений в искусственных и естественных зрительных системах:

а – проецирование изображения объекта на сетчатку глаза;

б – проецирование в фотокамере при параллельном расположении поверхности объекта и плоскости проецирования; в – проецирование в фотокамере при непараллельном расположении поверхности объекта и плоскости проецирования.

1.3. Геометрическое преобразование, описанное выше в п. 1.2, еще более усложняется, если отображаемый объект приближается к объективу камеры (хрусталику глаза) на дистанцию, сопоставимую с его фокусным расстоянием (рис. 1.в). Этот случай соответствует проективному геометрическому преобразованию изображения, описываемому формулами (2), включающими два дополнительных параметра a_7 и a_8. Однако в большинстве практически значимых случаев наблюдаемые объекты находятся достаточно далеко от объектива камеры, при этом значения параметров a_7 и a_8 стремятся к нулю, что приводит формулы (2) к виду (1).

$$\begin{cases} x' = (a_1x + a_2y + a_3)/(a_7x + a_8y + 1) \\ y' = (a_4x + a_5y + a_6)/(a_7x + a_8y + 1) \end{cases}.$$

1.4. Как указывалось выше, аффинное преобразование можно представить в виде последовательности более простых преобразований:

$$\mathbf{A} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{pmatrix} = \mathbf{A}_1\mathbf{A}_2\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5\mathbf{A}_6,$$

где $\mathbf{A}_1$ соответствует линейному смещению центра тяжести изображения вдоль оси абсцисс, $\mathbf{A}_2$ соответствует смещению центра тяжести изображения вдоль оси ординат, $\mathbf{A}_3$ соответствует сжатию вдоль некоторого направления (см. п.1.2), $\mathbf{A}_4$ соответствует зеркальному отражению, $\mathbf{A}_5$ соответствует масштабированию, равномерному по двум декартовым координатам, $\mathbf{A}_6$ соответствует вращению объекта в плоскости изображения относительно начала координат. В общем случае при данном разложении аффинного преобразования порядок его составляющих имеет существенное значение. Однако операции $\mathbf{A}_1$ и $\mathbf{A}_2$ ортогональны и могут быть переставлены друг с другом местами. Операции $\mathbf{A}_5$ и $\mathbf{A}_6$ также ортогональны и могут быть переставлены друг с другом местами. Операция зеркального отражения $\mathbf{A}_4$ может выполняться независимо от операции $\mathbf{A}_3$ – до нее или после нее – а также до или после операции $\mathbf{A}_5$ (она ортогональна по отношению к операции $\mathbf{A}_5$). Однако выполнению зеркального отражения перед выполнением операции вращения $\mathbf{A}_6$, или после нее будут соответствовать разные значения параметра вращения в разложении (3). Пусть для определенности операция зеркального отражения выполняется первой (до операции $\mathbf{A}_6$).

1.5. Перед началом распознавания изображений (для обеспечения успеха операции распознавания) целесообразно, если это представляется возможным, измерить параметры его аффинного преобразования относительно его эталонного состояния и компенсировать это преобразование. В соответствии с приведенным в п. 1.4 разложением аффинного преобразования измерение и компенсация составляющих его простых преобразований должны выполняться в следующей последовательности: $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_4$, $\mathbf{A}_5$, $\mathbf{A}_6$. Тогда, если

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3 \mathbf{A}_4 \mathbf{A}_5 \mathbf{A}_6 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix},$$

в результате компенсации получим

$$\begin{pmatrix} x'' \\ y'' \\ 1 \end{pmatrix} = \mathbf{A}_6^{-1} \mathbf{A}_5^{-1} \mathbf{A}_4^{-1} \mathbf{A}_3^{-1} \mathbf{A}_2^{-1} \mathbf{A}_1^{-1} \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} =$$

$$= \mathbf{A}_6^{-1} \mathbf{A}_5^{-1} \mathbf{A}_4^{-1} \mathbf{A}_3^{-1} \mathbf{A}_2^{-1} \mathbf{A}_1^{-1} \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3 \mathbf{A}_4 \mathbf{A}_5 \mathbf{A}_6 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \equiv \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}.$$

приведенных ниже материалах предлагается для исследования метод измерения параметров аффинного преобразования, основанный на сравнении изображения объекта с обобщенными эталонами [1, 2]. Как было сказано в [1, 2] и выше в п. 1.4, эталонным состоянием объекта при таком способе разложения его геометрического преобразования на простые составляющие является такое его пространственное положение, при котором центр тяжести его изображения расположен в начале координат.

1.6. Таким образом, для выполнения операций $\mathbf{A}_1^{-1}$ и $\mathbf{A}_2^{-1}$, необходимо вычислить абсциссу и ординату (x_c и y_c) центра тяжести изображения объекта как

$$x_c = \frac{\sum\limits_{x,y\in\Omega} x \cdot T(x,y)}{\sum\limits_{x,y\in\Omega} T(x,y)}, \qquad y_c = \frac{\sum\limits_{x,y\in\Omega} y \cdot T(x,y)}{\sum\limits_{x,y\in\Omega} T(x,y)},$$

и переместить центр тяжести в начало координат:

$$x^* = x - x_c, \quad y^* = y - y_c,$$

где Ω - двумерная область определения изображения объекта, $T(x,y)$ – яркость пиксела с координатами $\{x, y\}$. Т.е. все пикселы, принадлежащие изображению объекта, перемещаются на $-x_c$ и на $-y_c$, соответственно.

в ходе дальнейших вычислений вместо реального перемещения пикселов можно учесть, что координаты x и y каждого пиксела, подставляемые в ниже используемые формулы, должны быть представлены как $x - x_c$ и $y - y_c$.

1.7. В соответствии с [1, 2] направление Θ и величину μ сжатия изображения, центрированного согласно пункту 1.6, можно вычислить с использованием следующих выражений:

$$\mu = \pm\sqrt{\frac{D+\sqrt{C^2+B^2}}{D-\sqrt{C^2+B^2}}},$$

$$\Theta = 0{,}5\ \mathrm{arctg}(C/B) + \pi n,$$

$$B = \iint_{\Omega} T(r,\varphi)\, r^2 \cos(2\varphi)\, dr d\varphi,$$

$$C = \iint_{\Omega} T(r,\varphi)\, r^2 \sin(2\varphi)\, dr d\varphi,$$

$$D = \iint_{\Omega} T(r,\varphi)\, r^2 dr d\varphi,$$

где $T(r,\varphi)$ – яркость пиксела изображения объекта, представленного в полярной системе координат, Ω – область определения изображения объекта в полярной системе координат $\{r, \varphi\}$ – соответственно радиус и азимут пиксела в полярной системе координат.

Используя тригонометрические формулы $\cos(2\varphi)=\cos^2(\varphi)-\sin^2(\varphi)$ и $\sin(2\varphi)=2\cos(\varphi)\sin(\varphi)$ и переходя от интегралов к дискретным суммам, также принимая во внимание, что теперь мы имеем дело с изображением объекта, центрированным согласно формулам (5), выражения (.8), (9) и (10) можно представить как

$$B = \sum_{x,y\in\Omega} T(x,y)[(x-x_c)^2 - (y-y_c)^2],$$

$$C = \sum_{x,y\in\Omega} T(x,y) 2(x-x_c)\,(y-y_c),$$

$$D = \sum_{x,y\in\Omega} T(x,y)[(x-x_c)^2 + (y-y_c)^2].$$

Подставив в формулы (6) и (7) значения B, C и D, вычисленные согласно формулам (11), (12) и (13), можно рассчитать направление R и величину μ сжатия центрированного изображения и, применив к нему масштабирование в 1/μ раз в направлении Θ, компенсировать преобразование $\mathbf{A}_3$, упомянутое выше в формуле (3). Такое

преобразование координат пикселов можно описать следующим выражением:

$$\begin{pmatrix} x^+ \\ y^+ \end{pmatrix} = \mathbf{R}(+\Theta)\mathbf{S}(1/\mu)\mathbf{R}(-\Theta)\begin{pmatrix} x^* \\ y^* \end{pmatrix},$$

где x^+ и y^+ – абсцисса и ордината пикселов центрированного изображения после компенсации его масштабирования с коэффициентом μ в направлении Θ, x^* и y^* – координаты пикселов центрированного изображения, $\mathbf{R}(-\Theta)$ – матрица вращения изображения на угол $-\Theta$ (поворачивает направление масштабирования к оси абсцисс), $\mathbf{S}(1/\mu)$ – матрица масштабирования в $1/\mu$ раз в направлении оси абсцисс, $\mathbf{R}(+\Theta)$ – матрица обратного вращения изображения на угол $+\Theta$.

$$\mathbf{R}(+\Theta) = \begin{pmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{pmatrix},$$

$$\mathbf{R}(-\Theta) = \begin{pmatrix} \cos(-\Theta) & -\sin(-\Theta) \\ \sin(-\Theta) & \cos(-\Theta) \end{pmatrix},$$

$$\mathbf{S}(1/\mu) = \begin{pmatrix} 1/\mu & 0 \\ 0 & 1 \end{pmatrix}.$$

Тогда

$$x^+ = {}^1\!/_\mu[(x - x_c)\cos(-\Theta) - (y - y_c)\sin(-\Theta)]\cos\Theta -$$
$$-[(x - x_c)\sin(-\Theta) + (y - y_c)\cos(-\Theta)]\sin\Theta,$$

$$y^+ = {}^1\!/_\mu[(x - x_c)\cos(-\Theta) - (y - y_c)\sin(-\Theta)]\sin\Theta +$$
$$+[(x - x_c)\sin(-\Theta) + (y - y_c)\cos(-\Theta)]\cos\Theta.$$

1.8. Теперь остаются не измеренными и не компенсированными преобразования $\mathbf{A}_5$, $\mathbf{A}_6$ и $\mathbf{A}_4$ равномерного масштабирования, вращения и зеркального отражения (см. выше в п. 1.5). Преобразование $\mathbf{A}_5$ не зависит от изменения углового положения пиксела и не влияет на $\mathbf{A}_6$ и $\mathbf{A}_4$, поэтому параметры $\mathbf{A}_5 = \begin{pmatrix} M & 0 \\ 0 & M \end{pmatrix}$ можно измерить по проекции изображения на радиус полярной системы координат. В соответствии с [1, 2] коэффициент M равномерного масштабирования изображения может быть измерен согласно формуле (3.16) по смещению центра тяжести его проекции на радиус полярной системы координат относительно эталонного положения K центра тяжести этой проекции,

если изображение было предварительно центрировано (см. выше в п. 1.6), и если его пространственно неравномерное масштабирование было компенсировано согласно п. 1.7.

$$M = \left\{ \sum_{r,\varphi\in\Omega} r \cdot T(r,\varphi) \Big/ K \cdot \sum_{r,\varphi\in\Omega} T(r,\varphi) \right\} = \left\{ \sum_{x,y\in\Omega} T(x,y) \cdot \sqrt{(x^+)^2 + (y^+)^2} \Big/ K \cdot \sum_{x,y\in\Omega} T(x,y) \right\}. \quad (3.16)$$

Примем для определенности K=10, тогда, подставив в (3.16) координаты x^+ и y^+ пикселов исходного изображения, полученные в результате компенсации преобразований $\mathbf{A}_1$, $\mathbf{A}_2$ и $\mathbf{A}_3$ по формулам (3.4), (3.5), (3.14) и (3.15), вычислим коэффициент M равномерного масштабирования изображения, что позволит затем компенсировать преобразование $\mathbf{A}_5$, выполнив операцию

$$\begin{pmatrix} x^{\sim} \\ y^{\sim} \end{pmatrix} = \begin{pmatrix} 1/M & 0 \\ 0 & 1/M \end{pmatrix} \begin{pmatrix} x^+ \\ y^+ \end{pmatrix}. \quad (3.17).$$

1.9. Остались не компенсированными преобразования $\mathbf{A}_6$ и $\mathbf{A}_4$ вращения и зеркального отражения. Эти преобразования влияют на равномерное изменение угловой координаты пикселов в полярной системе координат и не зависят от распределения пикселов по радиальной координате. Поэтому параметры этих преобразований могут быть измерены по проекции изображения на окружность. Однако общий подход к измерению параметров этих преобразований требует применения спектрального анализа [1, 2]. При этом не удастся оперировать только численными значениями координат пикселов, как это делалось в предыдущих пунктах. Вместо этого придется реально переводить изображение в полярную систему координат, проецировать пикселы изображения на окружность и использовать дискретное преобразование Фурье. Эти операции исследуются в лабораторных работах №2, №4 и №5, поэтому в настоящей работе измерение параметров преобразований $\mathbf{A}_6$ и $\mathbf{A}_4$ вращения и зеркального отражения производиться не будет.

В заключение необходимо отметить, что описанный выше в пунктах 1.6-1.8 математический аппарат оперирует с изображениями объектов, помещенными на нулевой фон, поэтому описанная методика измерения параметров и компенсации аффинного преобразования пригодна только в ситуациях, когда имеются средства опознавания на изображении пикселов, принадлежащих анализируемому объекту.

2 Порядок выполнения лабораторной работы

2.1. Получить у преподавателя изображения для выполнения лабораторной работы. Необходимо помнить, что для корректности выполнения выше описанных процедур измерения параметров аффинного преобразования изображения объекта, оно должно быть отделено от фона и других объектов; именно такие изображения объектов на нулевом фоне предлагаются студентам для экспериментов.

2.2. Измерить параметры аффинных преобразований этих изображений и нормализовать их относительно этих преобразований по методике, изложенной в разделе 1. Нормализующие геометрические преобразования с вычисленными параметрами можно выполнить, используя программу, написанную для лабораторной работы №2, или вращая и масштабируя изображения раздельно по осям x и y с помощью подходящих графических редакторов. Компенсацию выше рассмотренных компонентов аффинного преобразования можно также выполнить, пересчитывая координаты каждого пиксела по формулам (3.14), (3.15) и (3.17). Это будет не вполне корректно, т.к. при этом не будет выполняться интерполяция, в то время как пересчитанные координаты пикселов не будут целыми числами, и их придется округлять до целого, однако, для предложенного эталонного значения коэффициента равномерного масштабирования, соответствующего K=10, эта некорректность не скажется на внешнем виде нормализованных изображений.

2.3. Проанализировать и объяснить результаты нормализации изображений.

2.4. Представить полученные результаты преподавателю.

Лабораторная работа 2 Преобразование координат на плоскости

Цель работы: составить программу для преобразований двухмерного геометрического объекта на плоскости. Продемонстрировать результаты на экране компьютера.

Краткие теоретические сведения

В компьютерной графике широко используются геометрические преобразования, как двухмерные, так и трехмерные. Необходимость в преобразованиях возникает в различных ситуациях: при переходе от декартовых координат изображаемого объекта к физическим координатам устройства; при получении различиях видов, разрезов или сечений детали; при построении наглядных изображений объекта в перспективе или аксонометрии; при перекосе, масштабировании и повороте изображений. Все эти преобразования можно выполнить на основе математических методов.

При выполнении лабораторной работы решаются задачи формирования исходного изображения плоского объекта с использованием графического расширения алгоритмического языка высокого уровня (Pascal, C, C++, Delphi); программной реализации математических методов геометрических преобразований на плоскости.

Математические методы

Двухмерные преобразования

Перенос – смещение графических примитивов на один и тот же вектор. При перенесении изображения без поворота связь между перемещенной точкой и ее первоначальным положением выражается уравнениями (рис. 1):

$$x' = x + Dx, y' = y + Dy,$$

или в векторной форме:

$$[x'\ y'] = [xy] + [DxDy] \quad \text{или} \quad P' = P + T.$$

Масштабирование. Точки изображения можно промасштабировать (сжать или растянуть) в S_x раз вдоль оси X и в S_y раз вдоль оси Y, в результате получаются новые точки (рис.2):

$$x' = x \cdot S_x, \quad y' = y \cdot S_y .$$

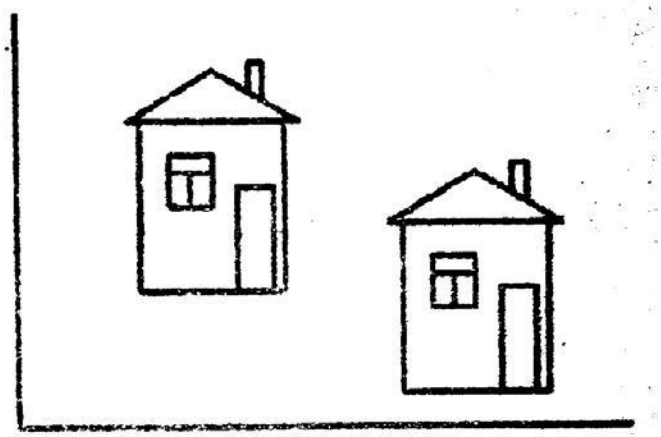


Рисунок 1 - Перенос объекта

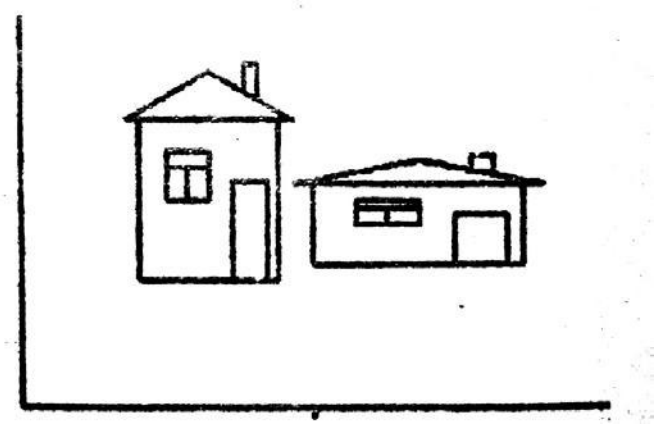


Рисунок 2 - Масштабирование объекта. $Sx = 1{,}5$, $Sy = 0{,}5$

Если определить S как $\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$, то можно записать в матричной форме:

$$[x'y'] = [xy] \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \quad \text{или } P' = P \cdot S.$$

На рисунке 2 показан домик, промасштабированный с коэффициентами 1,5 по оси x и 0,5 по оси y. Масштабирование производилось относительно центральной точки домика. Пропорции домика изменились и на рисунке 3.

В обоих случаях было применено неоднородное масштабирование, при котором $S_x \neq S_y$. Однородное масштабирование, для которого $S_x = S_y$, не влияет на пропорции.

Поворот. Точки могут быть повернуты на угол θ относительно начала координат, как показано на рисунке 4. Математически поворот определяется следующим образом:

$$x' = x \cdot \cos\theta - y\sin\theta,$$

$$y' = x \cdot \sin\theta + y\cos\theta,$$

в матричной форме:

$$[x'y'] = [xy]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

На рисунке 5 показан пример поворота изображения относительно центральной его точки на угол θ = 45°.

Однородные координаты

Описанные ранее преобразования переноса, масштабирования и поворота могут быть записаны в матричной форме:

$$P' = P + T,$$

$$P' = P \cdot S,$$

$$P' = P \cdot R.$$

Часто приходится сочетать эти элементарные виды преобразования по два и более раз. Между тем перенос реализуется с помощью сложения, а масштабирование и поворот – с помощью умножения. Для объединения этих преобразований целесообразно воспользоваться однородными координатами. С помощью таких координат все три преобразования можно реализовать при помощи умножения. Матричное произведение иногда называют композицией.

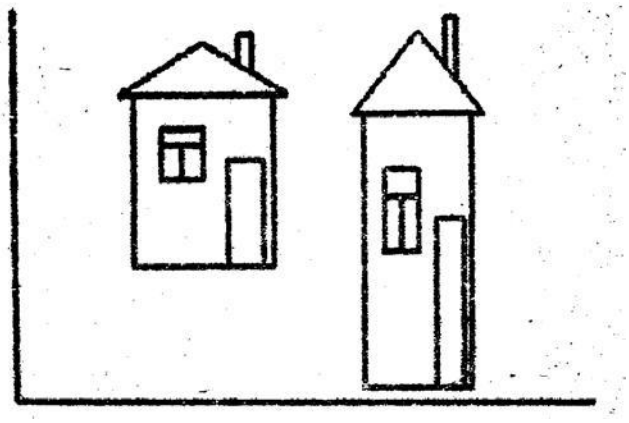


Рисунок 3 - Масштабирование объекта. Sx = 0,75, Sy = 1,6

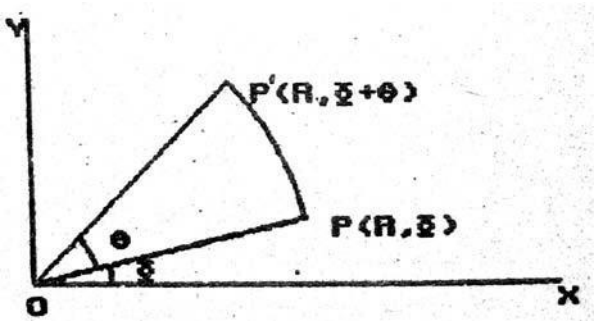


Рисунок 4 - Поворот точки

Точки в однородных координатах описываются трехэлементными вектор-строками. Чтобы получить другой вектор точки, матрица, на которую умножается вектор точки, должна иметь размер 3×3. Уравнения переноса записываются в виде матрицы преобразования однородных координат:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix} \qquad \text{или } P' = P \cdot T(D_x, D_y),$$

где

$$T(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}.$$

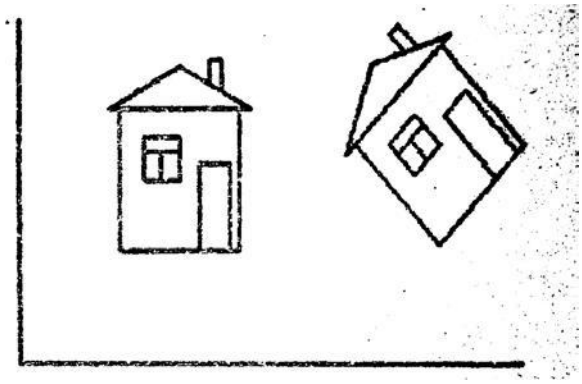


Рисунок 5 - Поворот объекта

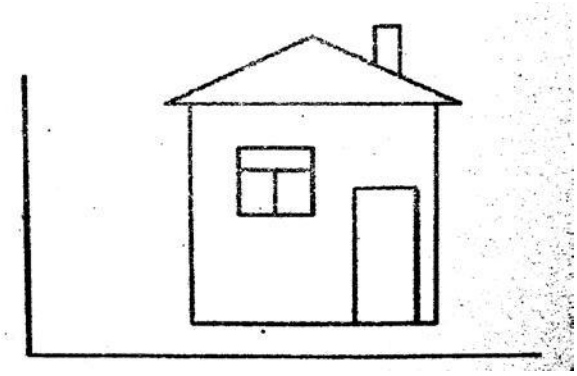


Рисунок 6 - Параметризация объекта

Запишем уравнения преобразования масштабирования в матричной форме:

$$[x'y'1]=[x \quad y \quad 1]\cdot\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Примем

$$S(S_x,S_y)=\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

и получим

$$P' = P \cdot S(S_x, S_y).$$

Уравнения преобразования поворота записываем в виде

$$[x' \quad y' \quad 1]=[x \quad y \quad 1]\cdot\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Принимая

$$R(\theta)=\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

окончательно получаем $P' = P \cdot R(\theta)$.

При повороте изображения вокруг произвольной точки M на угол θ разбиваем задачу на три элементарных преобразования:

1. Перенос точки M в начало координат.
2. Поворот вокруг начала координат.

3. Обратный перенос точки M в первоначальное положение. Эта последовательность показана на рисунке 5.

Первый перенос выполняется на $(-x_1, -y_1)$, а последующий – на (x_1, y_1) – обратный ему.

Преобразование композиции выглядит так:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_1 & y_1 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_1(1-\cos\theta)+y_1\sin\theta & y_1(1-\cos\theta)-x_1\sin\theta & 1 \end{bmatrix}.$$

Из примера видно, как применение однородных координат упрощает решение задачи на композицию двухмерных преобразований.

Порядок выполнения работы

Выполнение лабораторной работы можно разбить на три этапа:

1. Построение изображения с использованием базисных процедур и функций какого-либо алгоритмического языка.
2. Программная реализация какого-либо преобразования или цепочки (композиции) преобразований.
3. Оформление результатов работы.

Построение изображения . Исходный двухмерный объект может быть взят согласно варианту. Далее необходимо определить все опорные точки, задающие объект, как это показано на рисунке 6. Массив координат опорных точек и связи между ними есть геометрическая модель объекта.

Ниже приводится пример программы для построения домика, написанной на языке PASCAL. Комментарии в программе позволяют легко понять назначение операторов и процедур языка.

```
program home;

        uses Graph;  {ИСПОЛЬЗУЕМЫЕ МОДУЛИ}

  var

        Gd, Gm: Integer; {ОПИСАНИЕ ПЕРЕМЕННЫХ}
         x,y:array[1..23] of integer; {ОПИСАНИЕ МАССИВА
         ТОЧЕК}

  begin

        Gd:=Detect; InitGraph (Gd, Gm,' '); {ИНИЦИАЛИЗАЦИЯ}
```

```
if GraphResult () grOk then halt(1); { ГРАФИЧЕСКОГО
РЕЖИМА} <***************КРЫША*************>

Line(x[10],y[10],x[5],y[5]);  {ПРОЦЕДУРЫ ВЫЧЕРЧИВАНИЯ}
Line(x[5],y[5],x[11],y[11]);
Line(x[10],y[10],x[11],y[11]);

<****************ДОМ***************>

Line(x[1],y[1],x[2],y[2]);
Line(x[2],y[2],x[3],y[3]);
Line(x[3],y[3],x[4],y[4]);
Line(x[4],y[4],x[1],y[1]);

<***************ТРУБА**************>

Line(x[8],y[8],x[6],y[6]);
Line(x[6],y[6],x[7],y[7]);
Line(x[7],y[7],x[9],y[9]);

<****************ОКНО***************>

Line(x[12],y[12],x[14],y[14]);
Line(x[14],y[14],x[15],y[15]);
Line(x[15],y[15],x[17],y[17]);
Line(x[17],y[17],x[12],y[12]);
Line(x[18],y[18],x[19],y[19]);
Line(x[13],y[13],x[16],y[16]);

<***************ДВЕРЬ***************>

Line(x[20],y[20],x[21],y[21]);
Line(x[21],y[21],x[22],y[22]);
Line(x[22],y[22],x[23],y[23]);

CloseGraph;

End.
```

Преобразование изображения. Программу построения объекта необходимо теперь дополнить блоком, задающим геометрические преобразования. Для любого плоского объекта преобразование в однородных координатах в матричной форме можно записать в следующем виде:

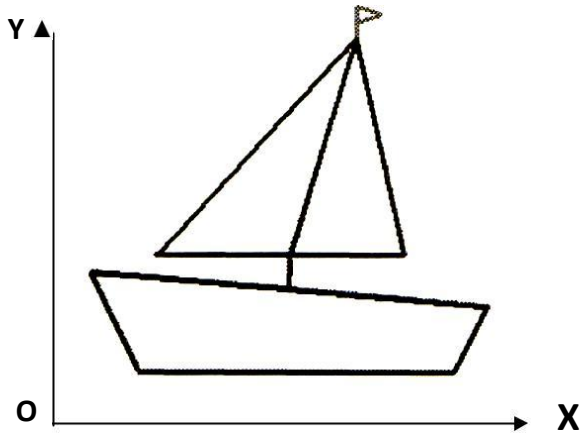
$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ . & . & . \\ x_n & y_n & 1 \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \\ x_3' & y_3' & 1 \\ . & . & . \\ x_n' & y_n' & 1 \end{bmatrix}$$

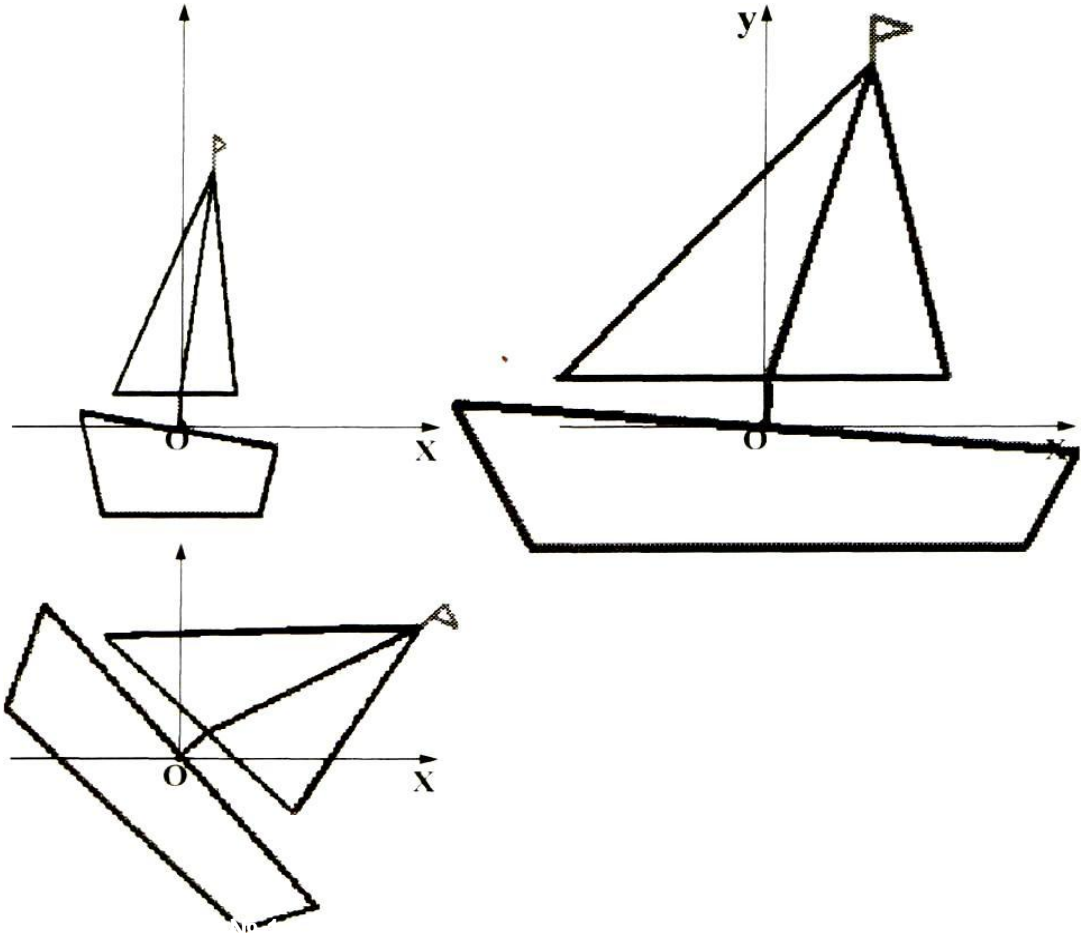
В этой записи первая матрица представляет собой массив координат точек, описывающих исходный объект. Матрица преобразований имеет размер 3×3. Выбирая необходимые значения элементов матрицы, можно реализовать то или иное геометрическое преобразование. Более того, цепочка (композиция) преобразований выполняется такой же операцией – умножением матрицы с исходными координатами точек на результирующую матрицу преобразований размером 3×3. В результате получается третья матрица с новыми координатами точек объекта.

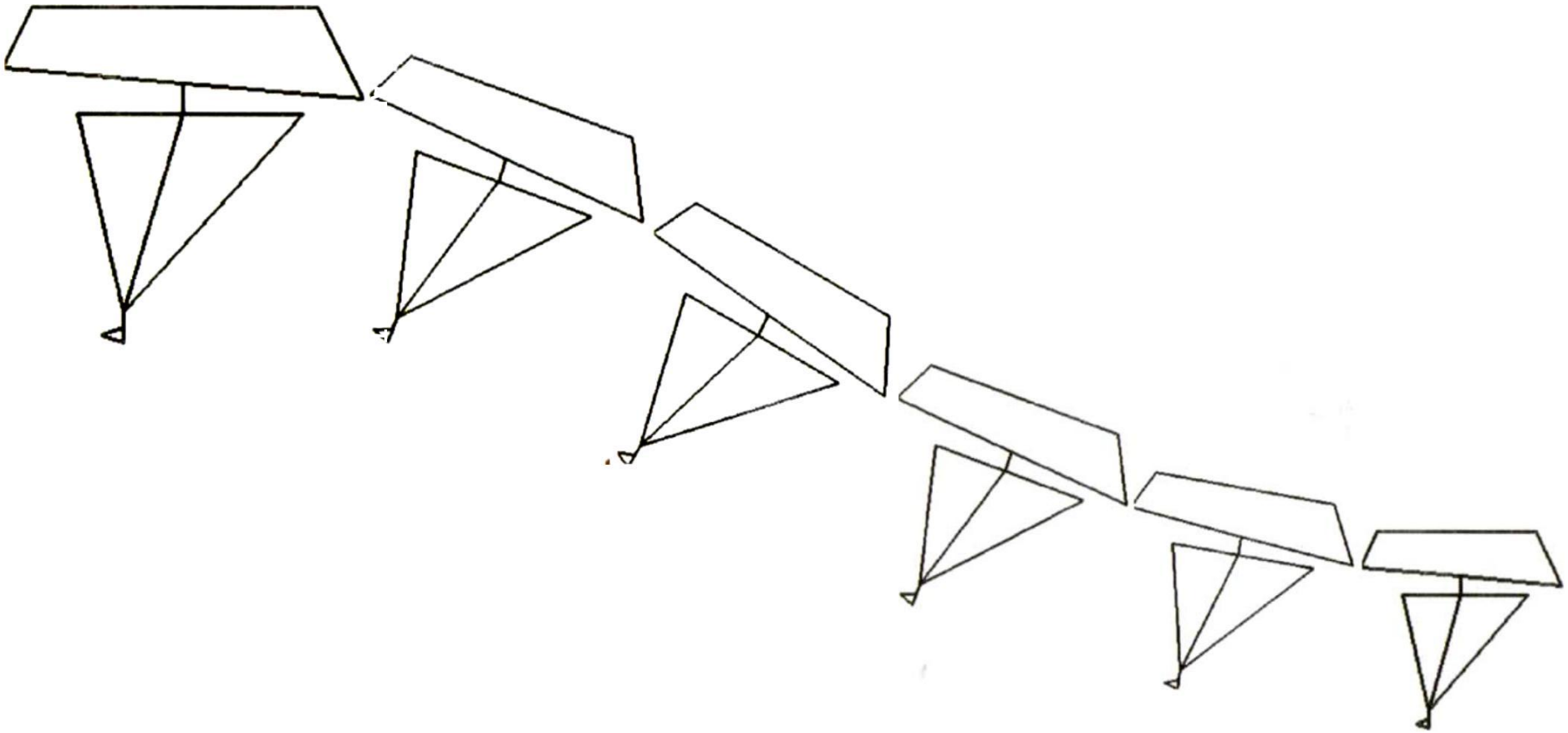
Таким образом, в основу блока геометрических преобразований положена процедура перемножения матриц. Ниже приводится пример такой процедуры, составленной на языке PASCAL.

```
procedure multiplication_mas(var
mas:point;mas1:matrix); var i:integer;
        x,y,z:real;
begin
        for i:=1 to NYZ do
        begin
{ ПЕРЕМНОЖЕНИЕ МАТРИЦ } { NYZ-КОЛИЧЕСТВО УЗЛОВ МАССИВА
РИСУНКА }
{МАТРИЦА КООРДИНАТ ТОЧЕК ‘MAS’ [NYZ,3] ПЕРЕМНОЖАЕТСЯ НА
МАТРИЦУ}
{                    ПРЕОБРАЗОВАНИЯ ‘MAS1’ [3.3]                    }
x:=(mas[i,1]*mas1[1,1])+ (mas[i,2]*mas1[2,1])+ (mas[i,3]*mas1[3,1]);
y:=(mas[i,1]*mas1[1,2])+ (mas[i,2]*mas1[2,2])+ (mas[i,3]*mas1[3,2]);
z:=(mas[i,1]*mas1[1,3])+ (mas[i,2]*mas1[2,3])+ (mas[i,3]*mas1[3,3]);
mas[i,1]:=x;
mas[i,2]:=y;
mas[i,3]:=z;
       end;
end;
```

Примеры преобразований







Листинг программы

```
#include <graphics.h>      // подключение необходимых библиотек
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#defme DIMX 12              // размерности матриц
#define DIMY 3              // постоянная перемещения //
#defmeSDVIG 10              постоянные масштабирования
#define PLUS 1.05
#define MINUS 0.95
#deflne a -3.14/18          // постоянная поворота - 10 градусов
double matkoor [DIMX] [DIMY],     // матрица коор-т об' екта
        center[3];              // матрица коор-т центра об'екта
int     InitMatrKoor();     // ф-ция иниц-ции матрицы коор-т
void    Actions(int choice),// ф-ция действий над об'ектом
                // choice - код нажатой клавиши
        InitCenter(),           // ф-ция иниц-ции матрицы коор-т центра об'екта
        Motion(int X,int Y,double m[DIM_X][DIM_Y]), // ф-ция смещения
                //X-смещ-ние по оси X,У -смещ-ние по оси У, m-мат-ца коор-т
        Rotation(double m[DIM_X][DIM_Y],double rad), // ф-ция поворота
                // m-мат-ца коор-т,rad-угол поворота
        Scaling(double,double,double m[DIM_X] [DIMYiy/ф-ция масшт-ния
        Multiplying(double m[DIM_X][DIM_Y],double m2[DIM_X][DIM_Y]),
        // ф-ция умнож-я матриц   (m на m2 с рез-ом в m)
Painter(double m[DIM_X][DIM_Y]); // ф-ция создания изоб-ния; int
i_see=0; int main( void) {
int gdrawiver = DETECT, gmode»errorcode; //запрос на автообнаружение
initgraph(&gdrawiver, &gmode, "..//bgi");     // инициализация графического ре-
жима
errorcode = graphresult();                    //чтение результата иниц-ии
  if (errorcode != grOk) {                          // ошибка произошла
      printf("Graphics error: %s\n",
      grapherrormsg(errorcode)); printf("Press any key to
      halt:");getch(); exit(l);
       if(InitMatrKoor()==0) exП(1);//если невожможно открыть файл -
       exit for(;;){
         cleardevice();                 //очистка экрана
         InitCenter();
         Painter(matkoor);              //вывод изоб-ния
         Actions(getch());              //действия
         closegraph();
           return (0);
        } intInitMatrKoor(){
```

```
    char string[80]; // имя файла с коор-ми
    int coor,ij;
    FILE *in;          // указатель на файл
        printf(MEnter filename with coordinates:");gets(string);
        if ((in = fopen(string, "rt")) = NULL){
        fprintf(stderr, "Cannot open input file.\n");
        return 0;
  for( i=O;i<DIM_X;i++)
  for(j=OJ<DIM_Y-1j++){fscanf(in,"%d",&coor);mat_koor[i][ =coor;}
  fclose(in);
  for(i=0;i<DIM_X;i++) mat_koor[i][2]=l;
  return 1;
  >id Painter(double m[DIM X][DIM Y]){
  line( m[0][0], m[0][1], m[1][0],m[1][1]);
  line(m[1][0], m[1][1], m[2][0],m[2][1]);
  line( m[2][0], m[2][1], m[3][0],m[3][1]);
  line( m[3][0], m[3][1], m[4][0],m[4][1]);
  line( m[4][0], m[4][1], m[0][0],m[0][l]);
  line( m[0][0], m[0][1], m[6][0],m[6][1]);
  line(m[5][0],m[5][1], m[7][0],m[7][1]);
  line(m[5][0],m[5][1], m[8][0],m[8][1]);
  line(m[6][0],m[6][1], m[8][0],m[8][1];
  line(m[7][0],m[7][1], m[8][0], m[8][1]);
  line( m[8][0], m[8][1], m[9][0],m m[9][1]);
  line(m[9][0],m[9][1], m[10][0],m[10][l]);
  line(m[10][0],m[10][l],m[11][0],m[11][1]);
  line(m[8][0],m[8]][1], m[11][0], m[11][1]);
     void Actions(int
choice)} switch(choice) {
        case 72 :Motion(0,-SDVIG,mat_koor);break;    //вверх- клавиша
  «UP» case 80 :Motion(0, SDVIG,mat_koor);break;    //вниз- клавиша
  «DOWN» case 75 :Motion(-SDVIG,0,mat_koor);break;//ВпраВО- клавиша
  «RIGHT» case 77 :Motion( SDVIG,0,mat_koor);break;   //влево- клавиша
  «LEFT» case 71 :Motion(-SDVIG,-SDVIG,mat_koor);break;//ВВерх и
  влево-
                                                        //клавиша «HOME»
        case 73 :Motion(SDVIG,-SDVIG,mat_koor);break; // вниз и
  влево«END» case 79 :Motion(-SDVIG, SDVIG,mat_koor);break;//ВВерх и
  вправо-
                                                        //клавиша «PGUP»
   case 81:Motion(SDVIG,SDVIG,mat_koor);break;//ВНН3 и Впраео«PGDN»
  case 43 :Scaling (PLUS, PLUS, mat_koor);break;//увеличение-
  «GRAY+»
  case 45 :Scaling(MINUS,MINUS,rnat_koor);break; //уменьшение - «-»
```

```
        case 47 : // поворот по часовой -клавиша «GRAY *»
        Motion(-center[0],-center[l], таг_коог);//перенос в центр коор-т
        Rotation(mat_koor, a);                          //поворот
         Motion(center[0],center[l], matkoor);// перенос в прежнее положение
                i_see++;if(i_see==36) i_see=0/отслеживаем общий угол пов-та
                break; case 42 :// поворот против часовой -клавиша
        «GRAY /»
                Motion(-center[0],-center [I ], matkoor);
                Rotation(mat_koor, -a);
                Motion(center[0],center[l ], matkoor);
                i_see--;if(i_see=-18) i_see= 18;
                break;
        case 119: Scaling(l,PLUS,mat_koor);break;//растяжение по оси X- «W»
        case 101:Scaling(MINUS,l,rnat_koor);break; // сжатие по оси X - «E»
        case 113:Scaling(l,PLUS,mat_koor);break; //растяжение по оси Y- «Q»
        case 97:Scaling(l,MINUS,mat_koor);break; // сжатие по оси Y- «A»
        case 27:exit(l);                        //выход из программы - клавиша «ESC»
        default:break;
    }

void InitCenter(){ center [0]=mat_koor [0] [0] ;center [ 1 ]=mat_koor [0][2]
  ;center[2]=mat_koor [0] [2];

void Motion(int X,int Y, double m[DIM_X][DIM_Y]) {
double perenos[DIM_Y][DIM_Y]={l, 0, 0,
                                        0,1,0,
                                        X, Y, 1};    // матрица переноса
    Multiplying( m, регепо5);//умнож-ние мат-цы коор-т на матрицу переноса

void Rotation(double m[DIM_X][DIM_Y], double rad){
  double rotate[DIM_Y][DIM_Y]={ cos(rad),sin(rad),0,
                                        -sin(rad),cos(rad),0,
                                            0,      0,      1 };// матрица поворота
     Multiplying( m, rotate); //умнож-ние мат-цы коор-т на матрицу поворота
void Scaling(double MM1,double MM2, double m[DIM_X][DIM_Y]){
double scale[DIM_Y][DIM_Y]={MMl,    0,    0,
                                    0,     MM2,0,
                                      0,       0,     1 }; //мат-ца масштабирования
        Motion(-center[0],-center[l], m); //перенос в центр коор-т
        Rotation(mat_koor, -a*i_see); //поворот об'екта в исходное положение
        Multiplying (m,sca1e);//умнож-ние мат-цы коор-т на мат-цу масшт-ния
        Rotation(mat_koor, a*i_see);   // поворот об'екта в прежнее положение
        Motion( center[0], center[l], m); // перенос в прежнее положение
```

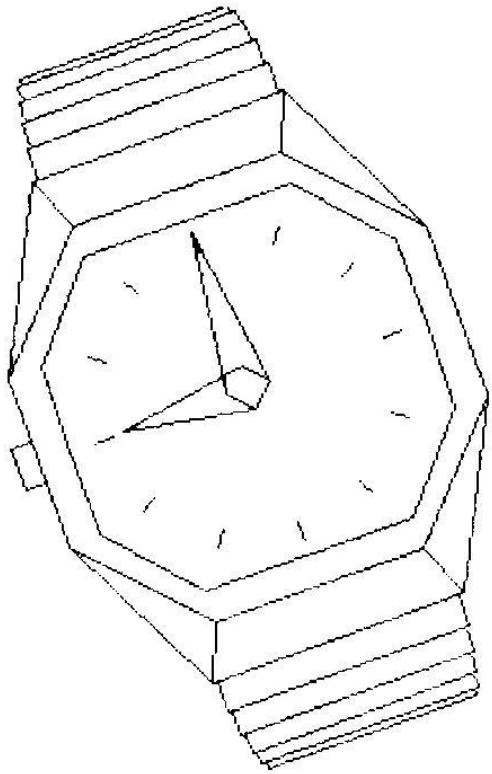
```
void Multiplying(double m[DIM_X][DIM_Y],double m2[DIM_X][DIM_Y]){
double mat_koor2 [DIMX] [DIMY];   //матрица обмена int comp_row,row,col;
//инициализация матрицы обмена
for(row=0;row<DIM_X;row++)for(col=0;col<DIM_Y;col++)
        mat_koor2[row][col]=0;
//непосредственное перемножение матриц m и m2 с рез-том в mat koor2
  for(row=0;row<DIM_X;row++)
     for(comp_row=0;comp_row<DIM_Y;comp_row++)
        for(col=0;col<DIM_Y;col++)
         mat_koor2[row][comp_row] += m[row][col]*m2[col][comp_row]; //обмен
содержимого матриц m и mat_koor2
         for(row=0;row<DIM_X;row++)
      for(col=0;col<DIM_Y;col++)m[row] [col]=rnat_koor2 [row] [col];
```

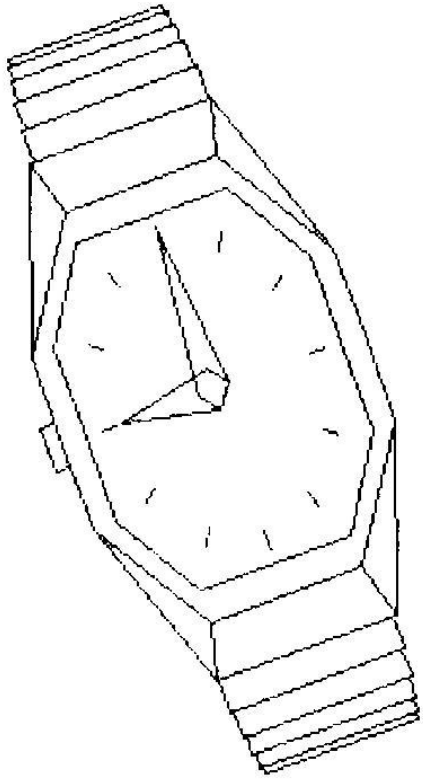
// знач-ния коор-т (поместите в какой-нибудь файл)

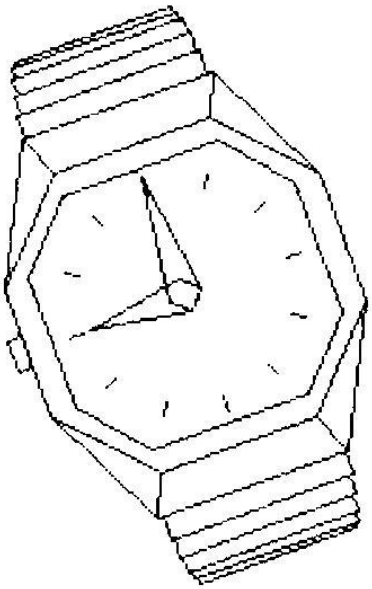
320 240
440 250
420 290
230 290
200 230
390 220
320 220
240 220
360 90
360 80
375 75
360 70

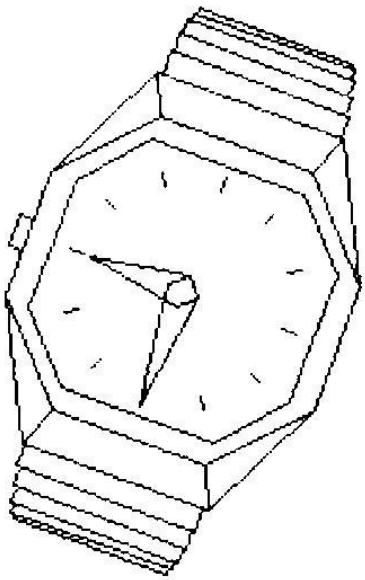
Варианты заданий

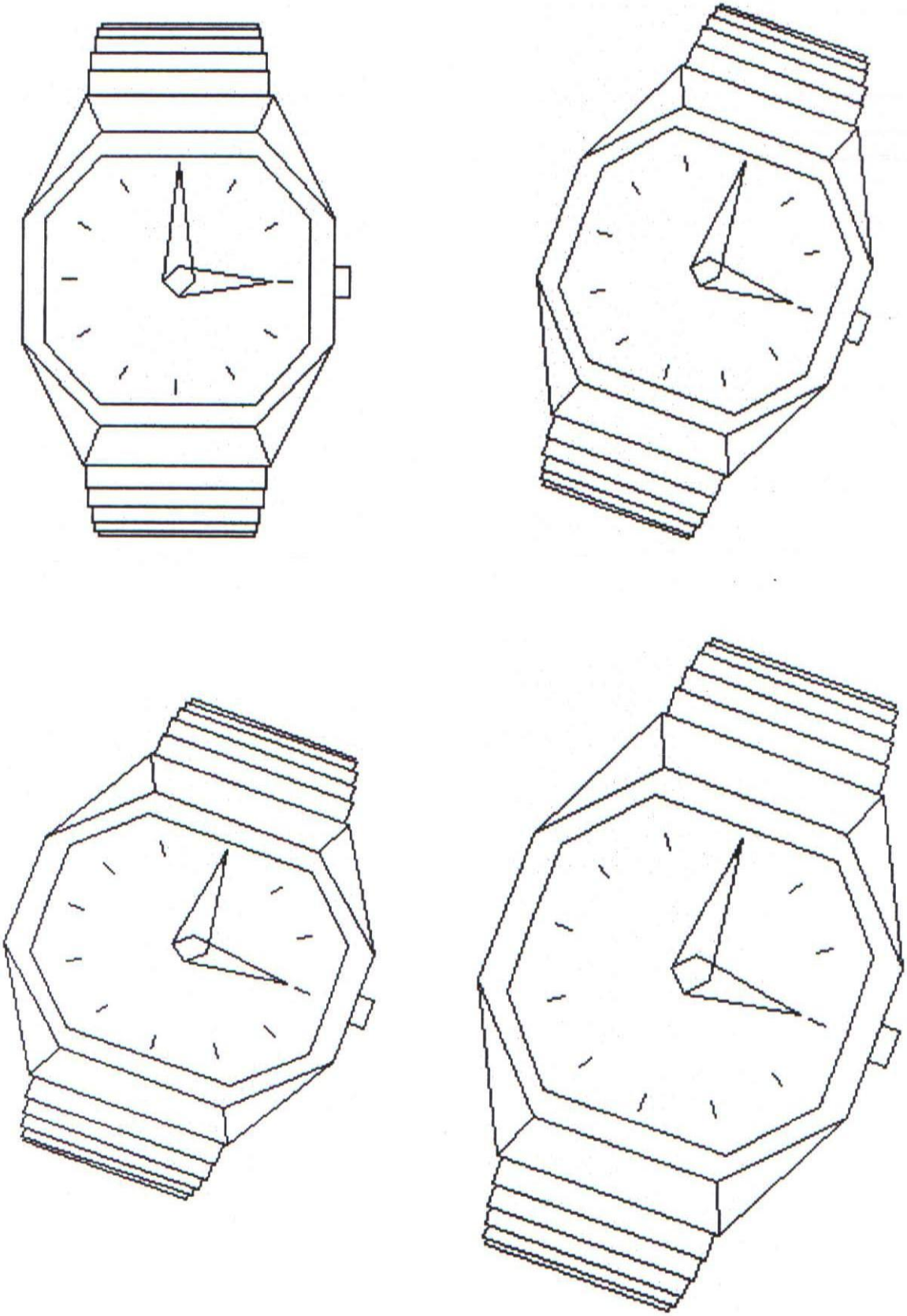
Вариант 1



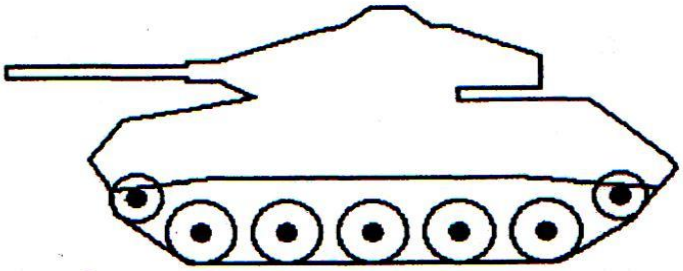


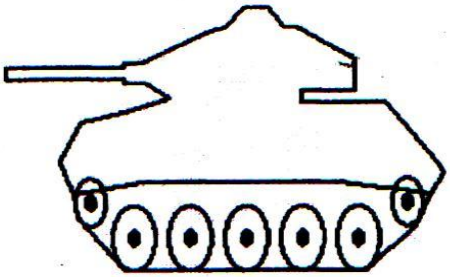


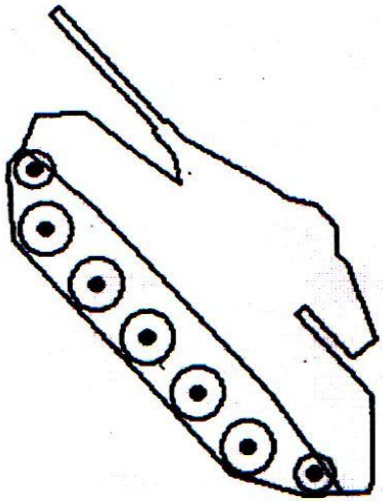


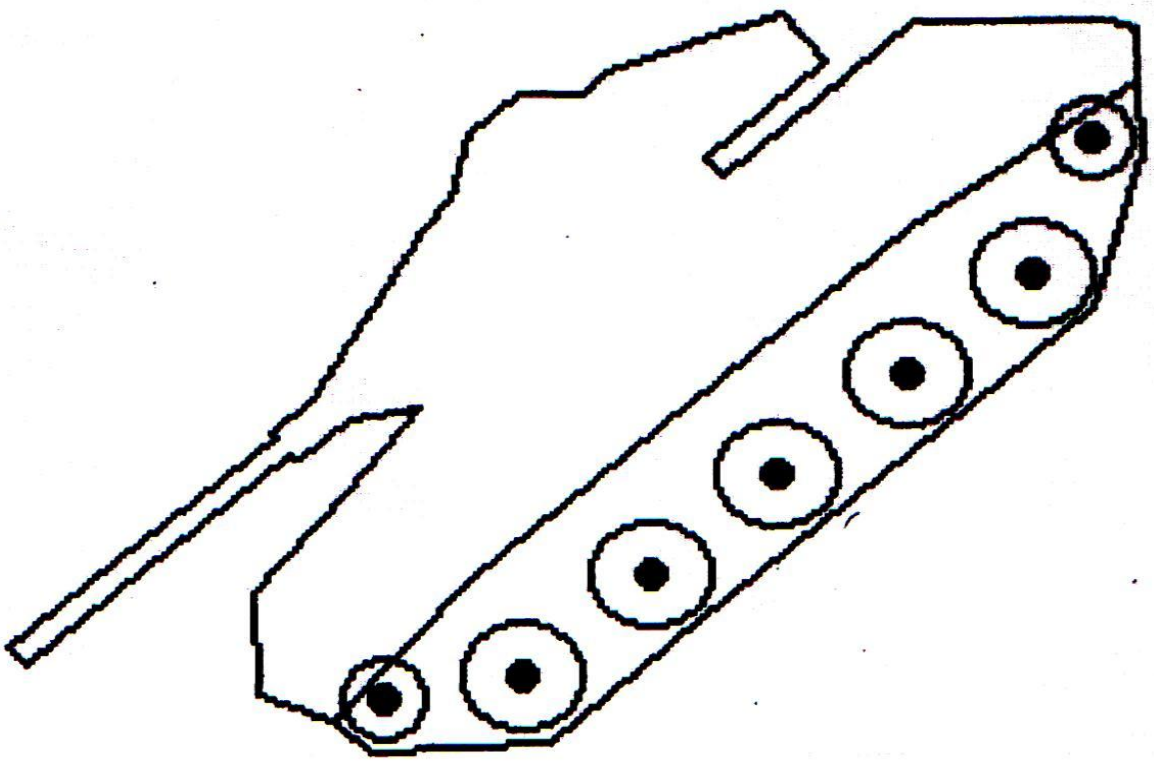


Вариант 2

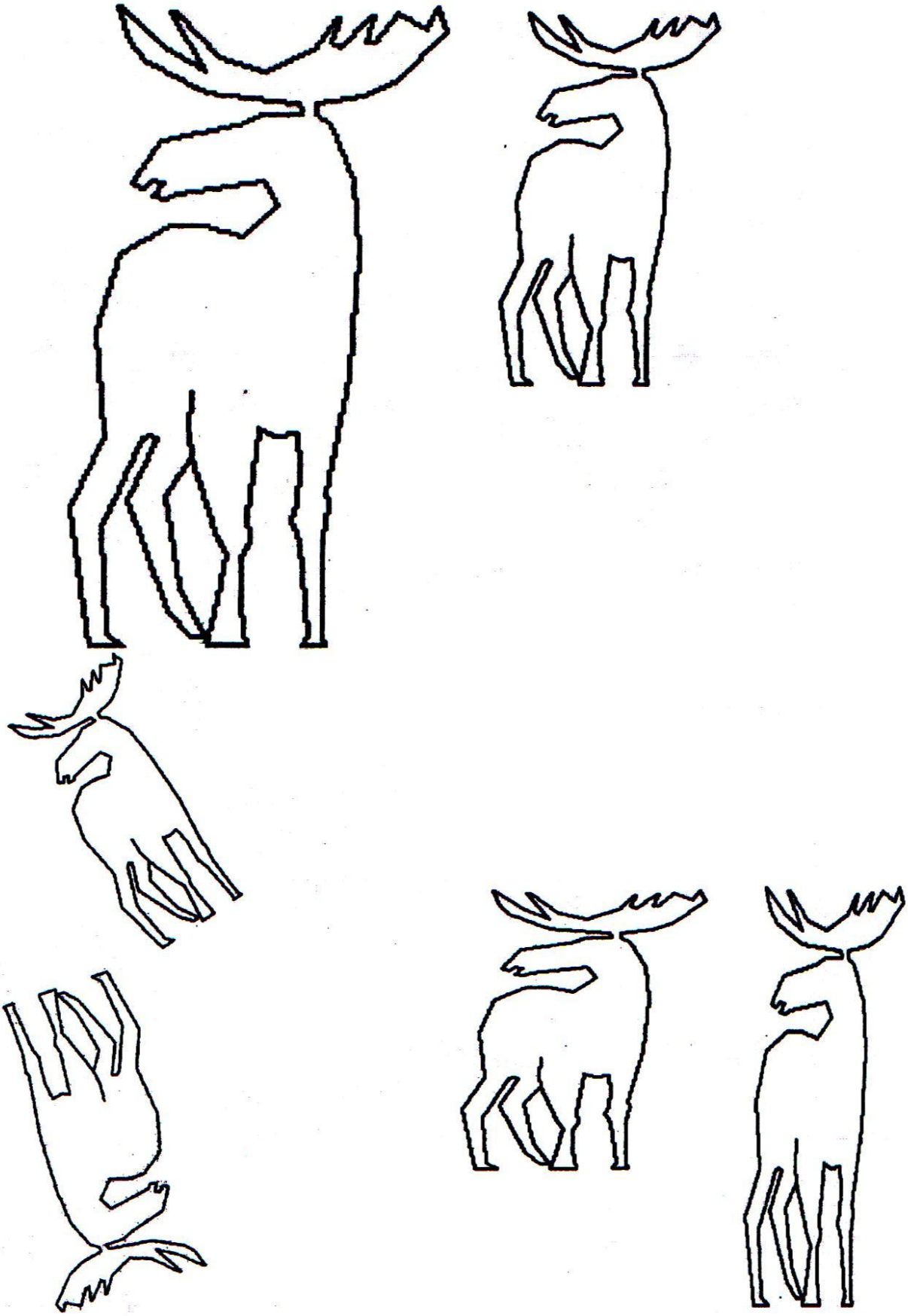




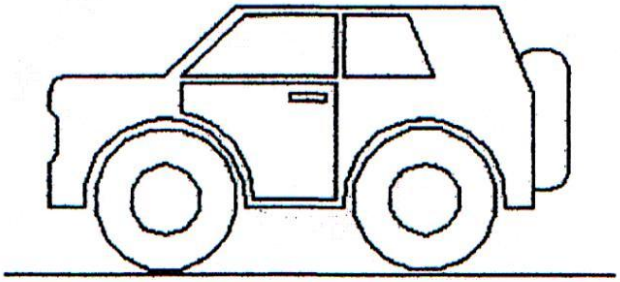


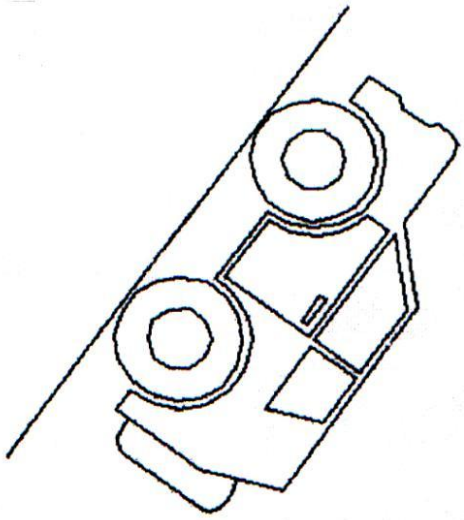


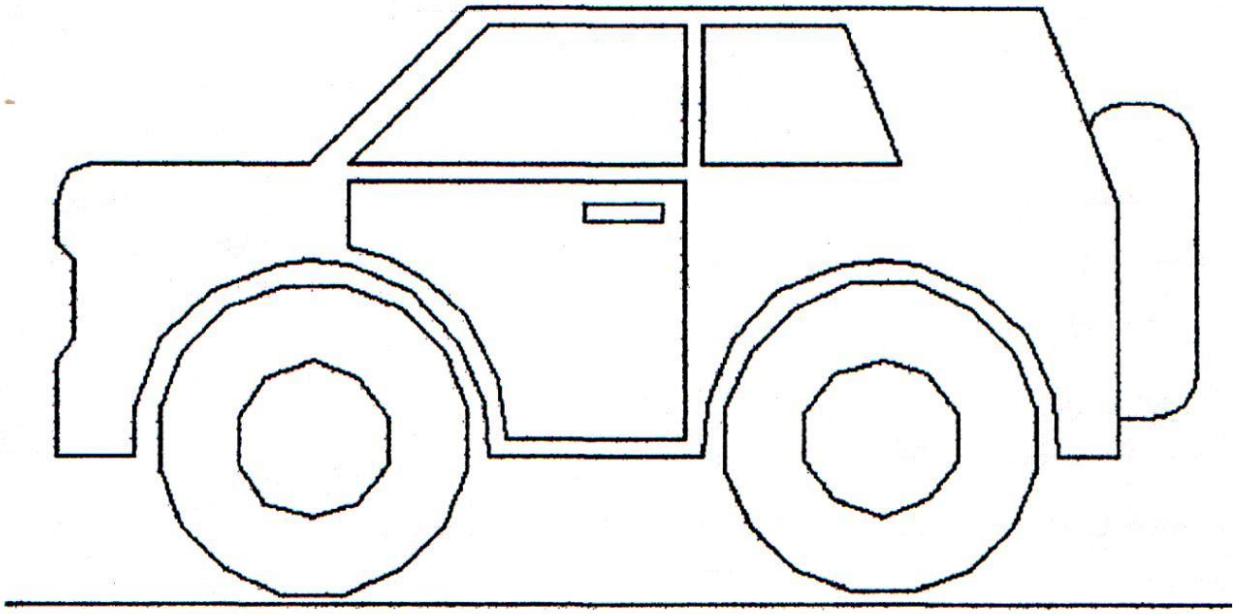
Вариант 3

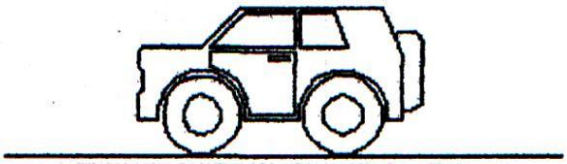


Вариант 4

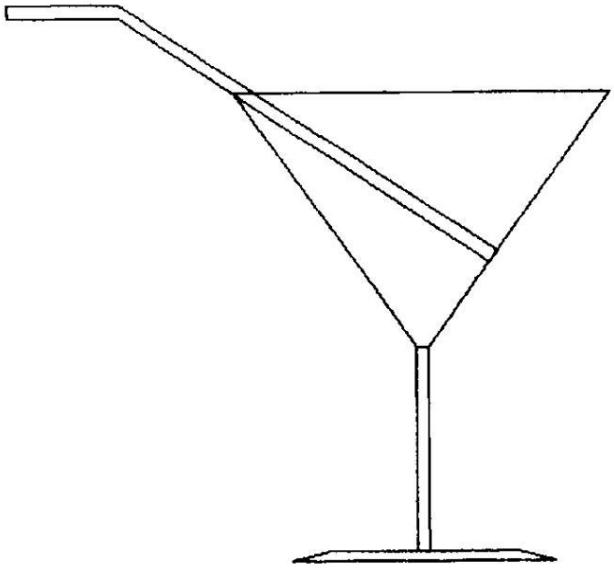


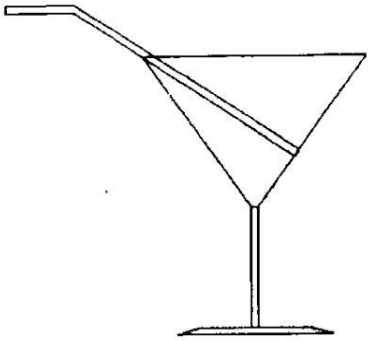


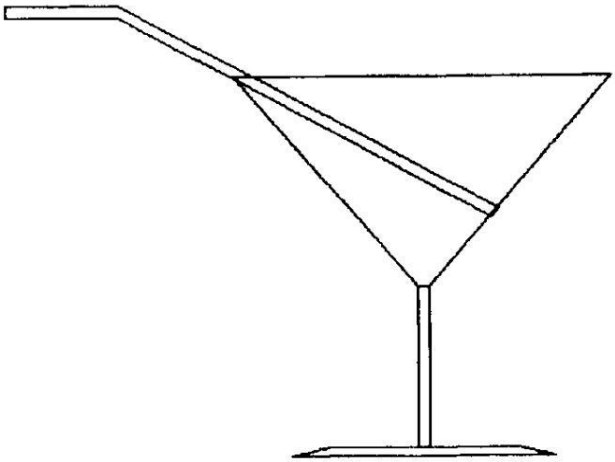


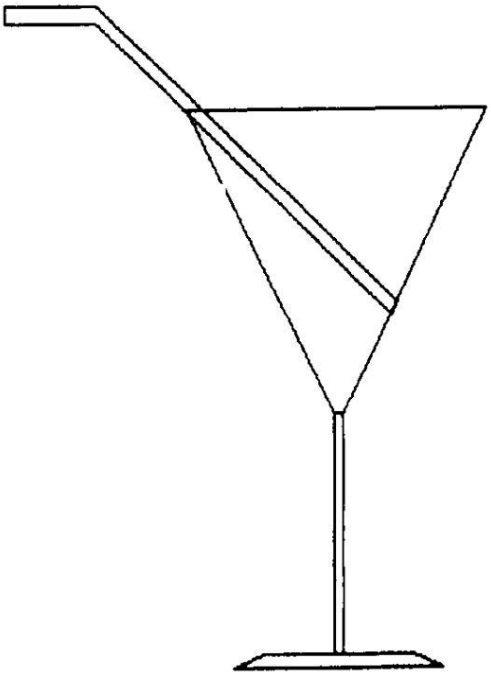


Вариант 5

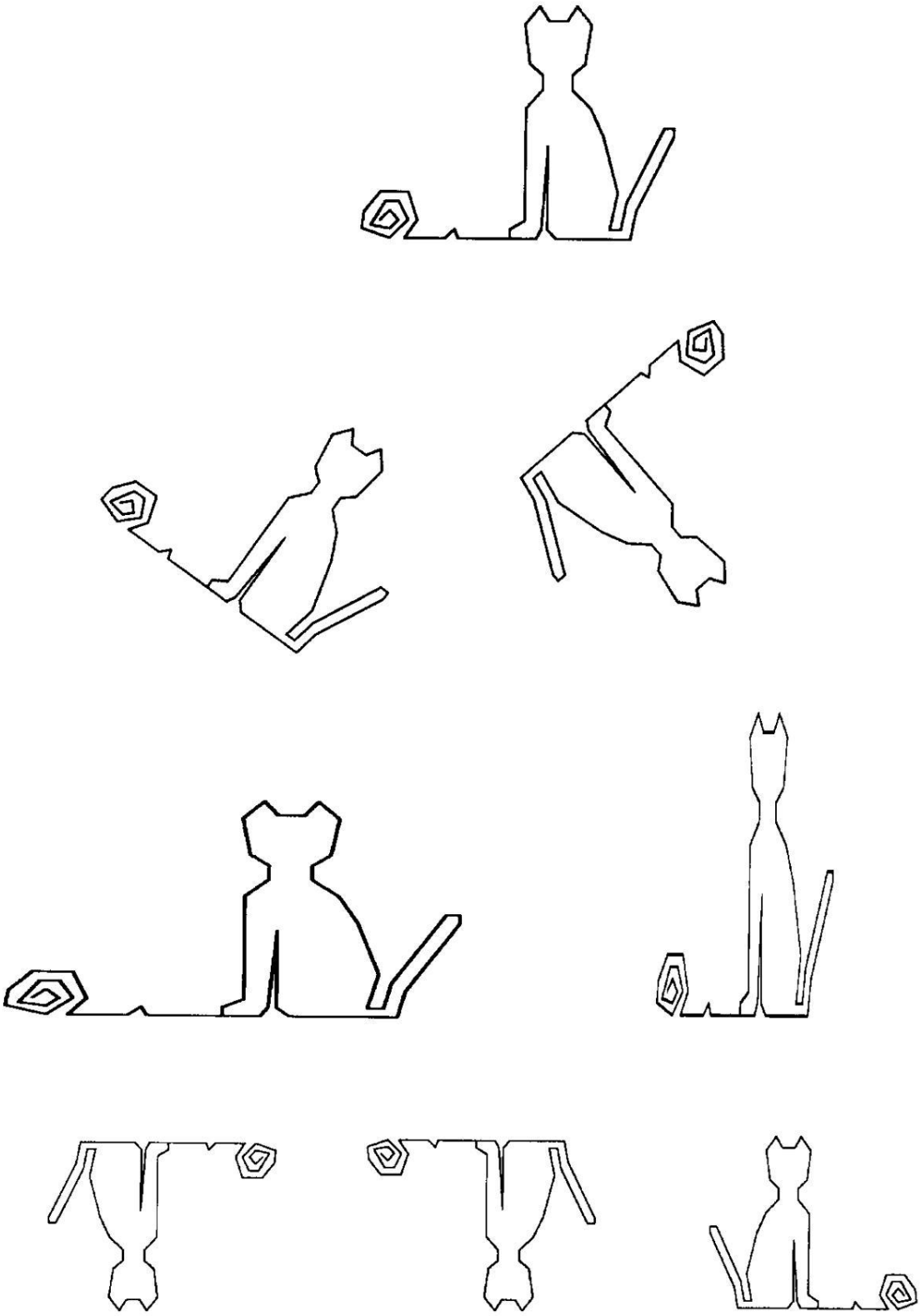








Вариант 6



Оформление отчета по лабораторной работе

В отчете должны быть представлены результаты всех этапов лабораторной работы:

1. Рисунок с указанием опорных точек объекта и их координат.
2. Краткое описание выполняемых геометрических преобразований в матричной форме.
3. Листинг программы, реализующей геометрические преобразования объекта.
4. Твердая копия графического экрана с результатом геометрических преобразований.

Лабораторная работа 3 Трехмерные преобразования

Цель работы: составить программу для трехмерных преобразований и вычисления плоских проекций трехмерных объектов. Продемонстрировать результаты на экране компьютера.

Краткие теоретические сведения

Трехмерные объекты чаще всего в графических документах задаются различными изображениями в виде нескольких ортогональных проекций (план, виды слева, справа спереди и т. д.). Но ортогональные проекции не обладают достаточной наглядностью и по ним сложно представить форму изображаемых объектов.

В отличие от ортогональных проекций проектируемых объектов компьютерные наглядные (перспективные, аксонометрические) изображения обладают реалистичностью, создают эффект объемности изображаемого пространства, позволяют оценить конструктивные и эстетические качества объектов проектирования.

Объектив фотоаппарата и глаз человека воспринимают окружающий нас трехмерный мир как перспективное изображение. Особенностью его является эффект перспективного укорачивания линейных размеров, которые изменяются обратно пропорционально удалению от центра проекций до объекта. Отсюда и недостатки перспективного изображения, которое, хотя и наиболее реалистично, но непригодно для непосредственного определения линейных размеров, площадей, углов.

Параллельная (аксонометрическая) проекция уступает в реалистичности, но выигрывает в возможности измерения линейных размеров, параллельных соответствующим координатным осям, так как искажение вдоль каждой из осей постоянно.

В технических чертежах применяют параллельные проекции, а центральные (перспективные) обычно используют художники и архитекторы для изображения общих планов. Черчение с применением перспективных проекций более или менее сложных объектов довольно трудоемко, но использование вычислительной техники позволяет упростить эту процедуру. С помощью вычислительных устройств удалось коренным образом изменить саму постановку проблемы. Средства компьютерной графики позволяют «видеть» объект в перспективе, легко исключая или представляя невидимые элементы изображения.

Перед разработчиками автоматизированных систем компьютерной графики в этом плане стоит следующая задача: представить реальный трехмерный объект на устройстве, имеющем двухмерную поверхность отображения (экран , чертеж). Решение ее состоит в получении проекции трехмерного объекта на плоскость.

Итак, различают перспективные и параллельные (ортогональные) проекции. В данной работе рассматривается математическое обеспечение и

предлагаются практические приемы трехмерных преобразований и вычисления плоских проекций трехмерного геометрического объекта.

Математические основы трехмерных преобразований и построения проекций

Обобщенная матрица преобразования 4×4 для трехмерных однородных координат имеет вид

$$T = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ h & i & j & r \\ l & m & n & s \end{bmatrix}$$

Эта матрица может быть представлена в виде четырех отдельных частей:

$$\left[\begin{array}{c|c} 3\times 3 & 3\times 1 \\ \hline 1\times 3 & 1\times 1 \end{array}\right].$$

Матрица 3×3 осуществляет линейное преобразование (в аффинной геометрии линейным называют преобразование, трансформирующее исходную линейную комбинацию векторов в некоторую также линейную их комбинацию) в виде изменения масштаба, сдвига и вращения. Матрица-строка 1×3 производит перенос, а матрица-столбец 3×1 – преобразование в перспективе. Последний скалярный элемент выполняет общее изменение масштаба. Полное преобразование, полученное путем воздействия на вектор положения матрицей 4×4 и нормализации преобразованного вектора, будем называть билинейным преобразованием. Оно обеспечивает выполнение комплекса операций сдвига, частичного изменения масштаба, вращения, отображения, переноса, а также изменения масштаба изображения в целом.

Точка в трёхмерном пространстве с учётом однородных координат представляется четырёхмерным вектором [x y z 1]. Диагональные элементы основной матрицы преобразования 4×4 выполняют частичное и полное изменение масштабов. Рассмотрим преобразование точки в трехмерном пространстве:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} ax & ey & jz & 1 \end{bmatrix} = \begin{bmatrix} x\times y\times z\times 1 \end{bmatrix},$$

которое производит частичное изменение масштабов. Общее изменение масштаба достигается за счет использования четвертого диагонального элемента:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix} = \begin{bmatrix} x \times y \times z \times 1 \end{bmatrix} = \begin{bmatrix} x \times y \times z \times 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{s} & \frac{y}{s} & \frac{z}{s} & 1 \end{bmatrix}.$$

Трехмерный сдвиг. Недиагональные элементы верхней левой подматрицы 3×3 от общей матрицы преобразования размера×**4** осуществляют сдвиг в трех измерениях:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ h & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x + yd + hz & bx + y + iz & cx + fy + z & 1 \end{bmatrix}.$$

Трехмерные вращения. Ранее было показано, что матрица ×3 обеспечивала комбинацию операций изменения масштаба и сдвига. Однако если определитель матрицы 3×3 равен +1, то имеет место чистое вращение около начала координат. Перед рассмотрением общего случая трехмерного вращения вокруг произвольной оси исследуем несколько частных случаев. При вращении вокруг оси x размеры вдоль этой оси x не изменяются. Таким образом, матрица преобразований будет иметь нули в первой строке и первом столбце, за исключением единицы на главной диагонали. Это приводит к матрице преобразования, соответствующей повороту на угол θ вокруг оси x и определяемой соотношением

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Пространственный перенос. Трехмерный линейный перенос изображения определяется выражением

$$\begin{bmatrix} X & Y & Z & H \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t & m & n & 1 \end{bmatrix}.$$

Ортографические проекции. Простая ортографическая проекция на нулевую плоскость, перпендикулярную любой из трех ортогональных осей, получается, когда соответствующий столбец в матрице преобразования размера 4×4 содержит все нули. Для проецирования на плоскости $x = 0$, $y = 0$, $z = 0$ используются следующие матрицы:

$$T_{\text{орт}}(x=0)=\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix},$$

$$T_{\text{орт}}(y=0)=\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix},$$

$$T_{\text{орт}}(z=0)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}.$$

Для ортографических проекций на плоскости $x = l$, $y = m$, $z = n$ необходимо применить еще преобразование переноса:

$$T_{\text{орт}}(z=n)=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&n&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&n&1\end{bmatrix}.$$

Аксонометрические проекции. Аксонометрическая проекция получается с помощью аффинного преобразования, определитель которого равен нулю. Существует несколько типов аксонометрических проекций, которые находят применение в начертательной геометрии и инженерной графике. Для получения математических зависимостей, описывающих аксонометрические проекции, используется матрица преобразования размера 4×4, необходимая для проведения аффинного преобразования системы точек. Точки затем проецируются на плоскость из центра проецирования в бесконечности.

Различают три вида проекций в зависимости от расположения плоскости проецирования и координатных осей:

1) изометрия – в плоскости проекций углы между каждой парой осей равны;
2) диметрия – в плоскости проекции равны между собой два угла между осями;
3) триметрия – в плоскости проекции все три угла между осями различны.

Изометрия, диметрия и триметрия получаются комбинацией поворотов, за которыми следует проекция из бесконечности. Если нужно описать проекцию на плоскость $z = 0$, то сначала необходимо осуществить поворот на угол Ф относительно оси y, затем на угол θ относительно оси x. В матричном виде эти преобразования имеют следующий вид:

$$[XYZH]=[x\,y\,z\,1]\times\begin{bmatrix}\cos\Phi & 0 & -\sin\Phi & 0\\ 0 & 1 & \theta\ 0\sin\theta & 0\\ \sin\Phi & 0 & \cos\Phi & 0\\ 0 & 0 & 0 & 0\end{bmatrix}0\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos & & \\ 0 & -\sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 0\end{bmatrix}=$$

$$=[x\,y\,z\,1]\begin{bmatrix}\cos\Phi & \sin\Phi\sin\theta & -\sin\Phi\cos\theta & 0\\ 0 & \cos\theta & \sin\theta & 0\\ \sin\Phi & -\cos\Phi\sin\theta & \cos\Phi\cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Перспективные преобразования. В матрице преобразования 4×4 ненулевые элементы в первых трех строках последнего столбца осуществляют перспективное преобразование. Перспективная проекция получается путем перспективного преобразования и проецирования на некоторую двухмерную плоскость «наблюдения». Перспективная проекция на плоскость $z = 0$ обеспечивается преобразованием

$$[XYZH]=[x\,y\,z\,1]\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & r\\ 0 & 0 & 0 & 1\end{bmatrix}=[x\ \ y\ \ 0\ \ (rz+1)].$$

Таким образом, чтобы получить перспективные изображения из произвольной точки наблюдения, вначале используют аффинные преобразования, позволяющие сформировать систему координат с осью z вдоль желаемой линии визирования. Затем применяют перспективное преобразование, а проекционное преобразование используют для того, чтобы спроецировать общее положение точек на плоскость наблюдения $z = 0$ в текущей системе координат.

Порядок выполнения работы

Выполнение лабораторной работы состоит из следующих этапов:

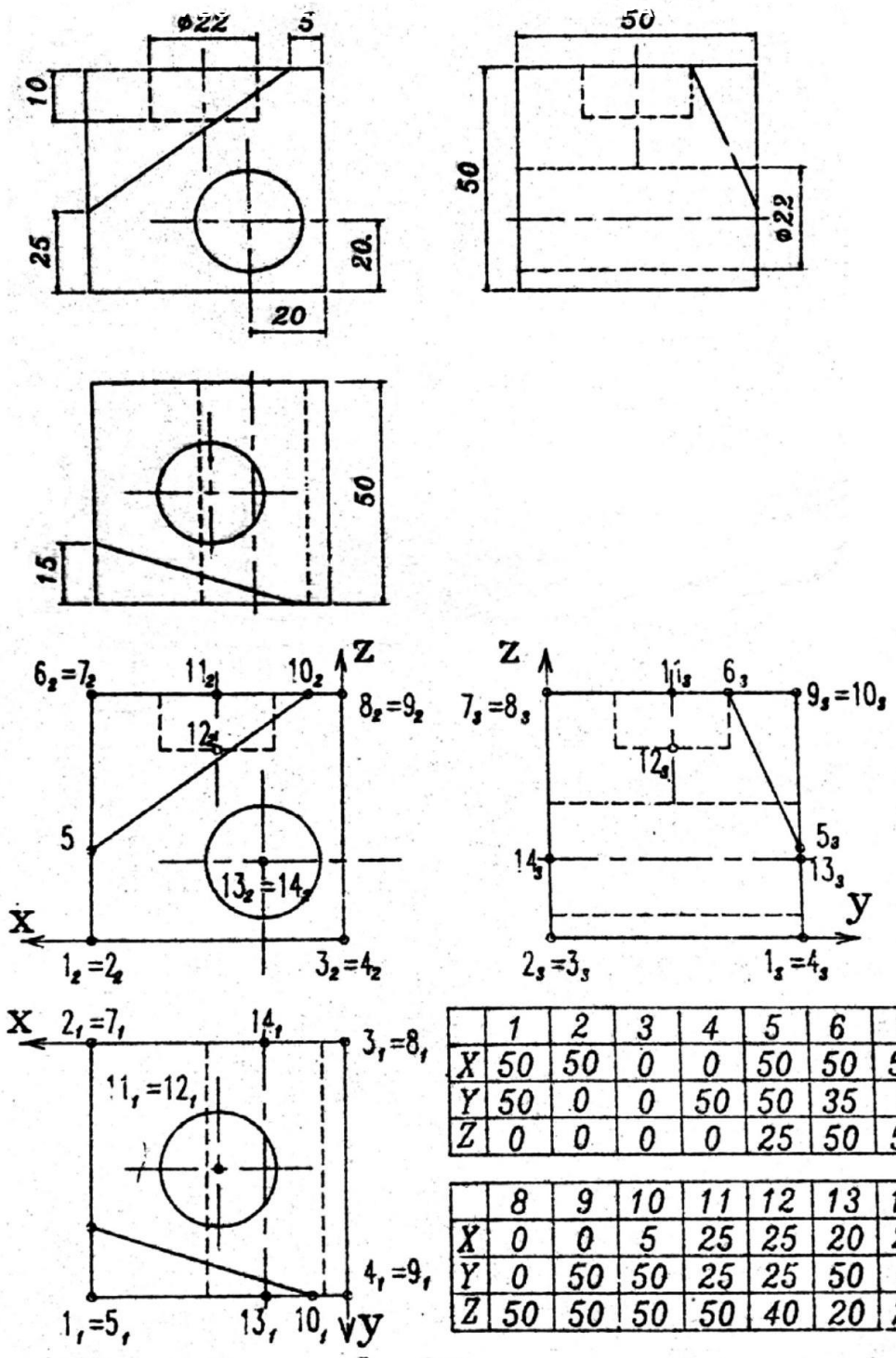
1) выполнение графической части – построение трех видов геометрического объекта в соответствии с выбранным вариантом. Для построения геометрической модели необходимо нанести все размеры. Для удобства строится аксонометрическая проекция (технический рисунок) объекта;
2) оцифровка изображения – обозначение узловых и опорных точек, определение их координат. Полученная информация сводится в таблицу;
3) формирование геометрической модели изображения – указание последовательности соединения узловых точек, диаметра и ориентации окружностей, центры которых располагаются в опорных точках;
4) составление программы, в результате выполнения которой на экране дисплея демонстрируются различные преобразования и проекции объекта.

Выполнение графической части работы

В соответствии с выбранным вариантом задания вычерчивают в карандаше три вида (рис. 1) геометрического тела (по заданным двум). Определяют размеры непосредственным измерением с учетом масштаба и наносят на чертеж согласно ГОСТ 2:307–68. В соответствии с привязкой системы координат выполняют аксонометрическое изображение заданного вида.

Оцифровка изображения

На чертеже обозначают цифрами узловые и опорные точки (см. рис. 1). Узловыми точками являются все вершины многогранника, опорными точками – центры окружностей, определяющих границы цилиндрических элементов поверхности. Далее следует определить числовые значения координат каждой из точек (в миллиметрах) и результаты свести в таблицу.



	1	2	3	4	5	6	7
X	50	50	0	0	50	50	50
Y	50	0	0	50	50	35	0
Z	0	0	0	0	25	50	50

	8	9	10	11	12	13	14
X	0	0	5	25	25	20	20
Y	0	50	50	25	25	50	0
Z	50	50	50	50	40	20	20

Рисунок 1 - Чертеж объекта. Координаты узловых и опорных точек

Формирование геометрической модели

Геометрическая модель объекта будет сформирована, если массив координат узловых точек дополнить информацией о последовательности связей этих точек, а массив координат опорных точек – информацией о диаметрах окружностей, центрами которых они являются, и указать ориентации в пространстве плоскости, в которой лежит каждая окружность (т. е. какой из координатных осей нормальна плоскость окружности).

Для рассматриваемого примера запись последовательности соединения узловых точек имеет вид:

А: 1 – 2 – 3 – 4 – 1
Б: 5 – 7 – 8 – 9 – 10
В: 6 – 10
Г: 1 – 5
Д: 2 – 7
У: 3 – 8
Ж: 4 – 9

Общее количество соединений должно быть равно количеству ребер.
Запись информации для опорных точек имеет вид:

T 11: (25, 25, 50), ∅22 – z
T 12: (25, 25, 40), ∅22 – z
T 13: (20, 50, 20), ∅22 – y
T 14: (20, 0, 20), ∅22 – y

Например, точка 11 с координатами x = 25, y = 25, z = 50 является центром окружности ∅22, плоскость которой перпендикулярна к оси z.

Разработка программы

На этом этапе вся полученная информация должна быть оформлена в виде графической программы. Пример программы на языке PASCAL для рассматриваемого геометрического тела приведен ниже. На рисунке 2 показан результат выполнения программы. Комментарии в программе позволяют определить назначение операторов, блоков и процедур.

Следует отметить, что в приведенном примере для построения искаженных в аксонометрии окружностей не используется процедура Ellipse, поскольку она не позволяет ориентировать эллипс с заданными углами наклона к направлениям осей экрана. В этом случае окружность задается массивом точек с помощью параметрических уравнений:

$$\begin{cases} x_0 = R\cos\theta, \\ y_0 = R\sin\theta, \end{cases}$$

где x_0 и y_0 – координаты текущей точки на окружности; R – радиус окружности.

Преобразование точек окружности выполняется таким же образом, как и преобразование узловых и опорных точек объекта. В результате получается эллипс с требуемыми параметрами для любой аксонометрической проекции.

```
program kub3;
uses            {используемые модули}
        Crt, Graph;
        type                    {описание типов переменных}
                a=array[1..300] of real;
                e=array[1..300] if integer;
        var     {-------Описание переменных-----------}
                x, y, z                         :a;
                U, V, z2 ,y2, x2                :e;
                i, t                            :integer;
                r1, r2, k, l, m, oy, ox, s      :real;
                Gd, Gm, Count                   :integer;
                coord                           :text;
        begin
                Gd := Detect;                           {Инициализация}
                InitGraph(Gd, Gm, 'd:\tpas');           {графического}
                if GraphResult ( ) grOk then Halt (1);  {режима}
        {=====================Тело Программы=====================}
        SetColor (Red);                 {Задание цвета линии}
        SetLineStyle (SolidLn, l, Blue);                {Задание типа линии}
        Assign (Coord, 'coord.txt');                    {Считывание}
        Reset (coord);                                  {данных}
        for i:= to 21 do
        ReadLn (Coord, x[i], y[i], z[i]);               {из}
        Close (Coord);                                  {файла}
                r1:=11; r2:=11; s:=0;
                        for i:=22 to 62 do              {задание 1 и 2 окружностей}
        begin
                s:=s+0.157;
                z[i]:=50;               z[i+41]:=40;
                x[i]:=(25+cos(s)*r1);   x[i+41]:=x[i];
                y[i]:=(25+sin(s)*r1);   y[i+41]:=y[i];
                        end;
                        for i:=104 to 144 do            {задание 3 и 4 окружностей}
        begin
                s:=s+0.157;
                y[i]:=50;               y[i+41]:=0;
                x[i]:=20+cos(s)*r2;     x[i+41]:=x[i];
                z[i]:=20+sin(s)*r2;     z[i+41]:=z[i];
        end;
                k:=0.523; l:=0.523                      {углы в радианах}
                        for i:=1 to 185 do              {адаптация координат к экрану}
        begin
                m:=0.3; ox:=300; oy:=250
                U[i]:=Round (ox+(y[i]*cos[k] - x[i]*cos(l))/m;
                V[i]:=Round (oy+(-z[i]+y[i]*sin(k)+x[i]*sin(l)))/m;
        end;
```

```
    for i:=11 to 30 do      {Рисование образующих 1-го цилиндра}
                    Line(U[2*i], V[2*i], U[2*i+41], V[2*i+41]);
    for i:=52 to 72 do      {Рисование образующих 2-го цилиндра}
                    Line(U[2*i], V[2*i], U[2*i+41], V[2*i+41]);
                    SetLineStyle(0,1,1);   {Задание типа линии}
    for i:=1 to 20 do       {Рисование ребер}
                    Line(U[i], V[i], U[i+1], V[i+1]);
    for i:=22 to 61 do      {Рисование окружности-1}
                    Line(U[i], V[i], U[i+1], V[i+1]);
    for i:=63 to 102 do     {Рисование окружности-2}
                    Line(U[i], V[i], U[i+1], V[i+1]);
    for i:=104 to 143 do    {Рисование окружности-3}
                    Line(U[i], V[i], U[i+1], V[i+1]);
    for i:=145 to 184 do    {Рисование окружности-4}
                    Line(U[i], V[i], U[i+1], V[i+1]);
    ReadLn;
    CloseGraph; { покинуть графический режим}
end.
```

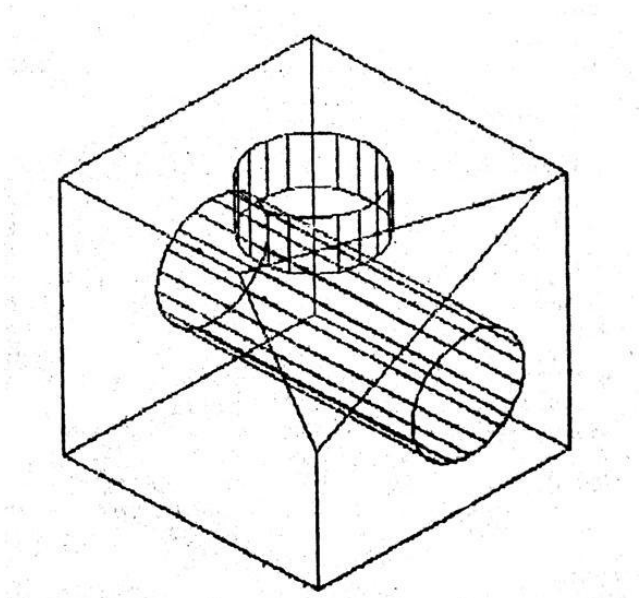


Рисунок 2 - Результат выполнения программы

Примеры преобразований

Примеры преобразований приведены на рисунках 3-5.

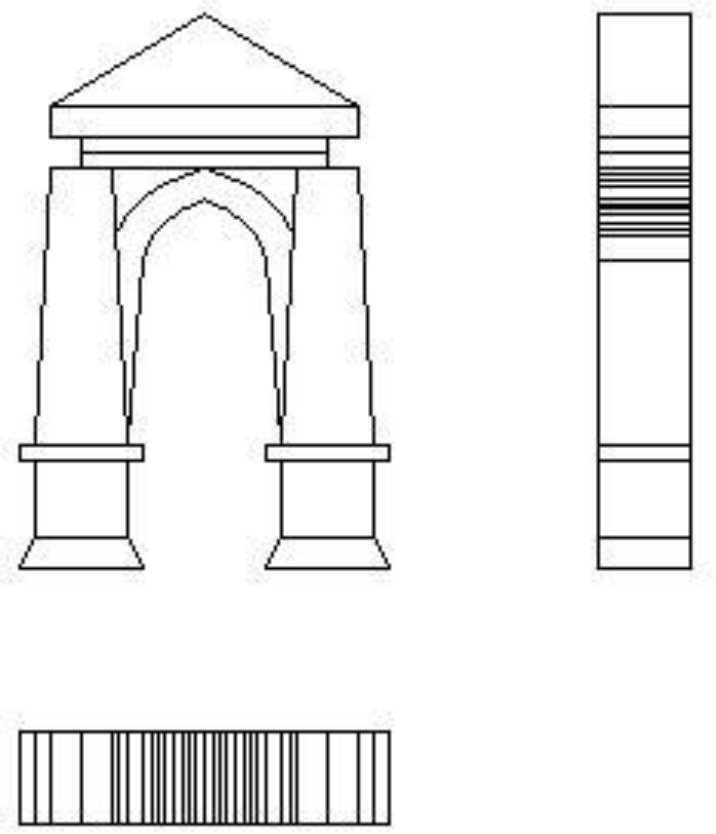


Рисунок 3 - Ортогональные проекции

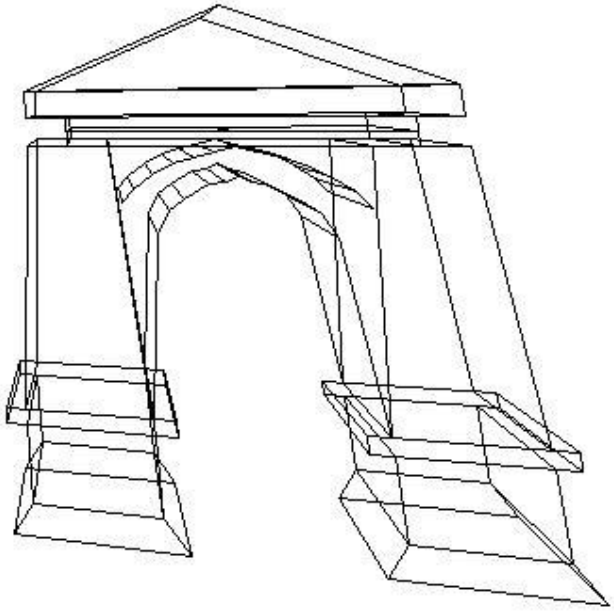
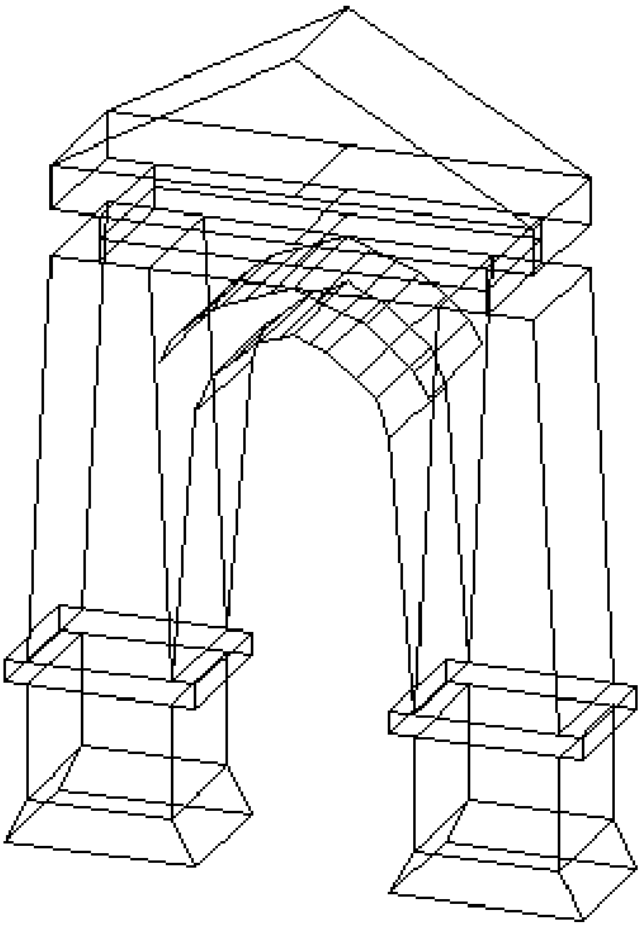


Рисунок 4 - Перспектива с одной точкой схода лучей



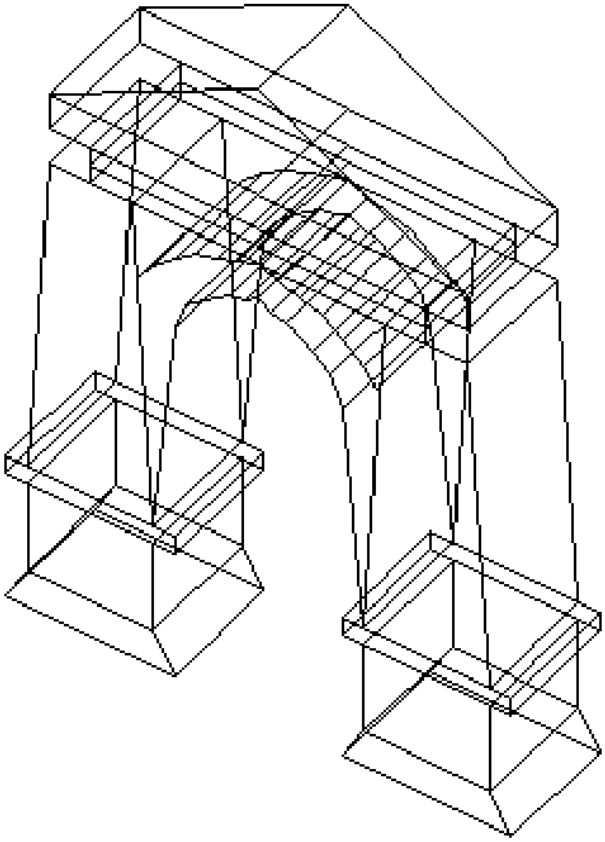
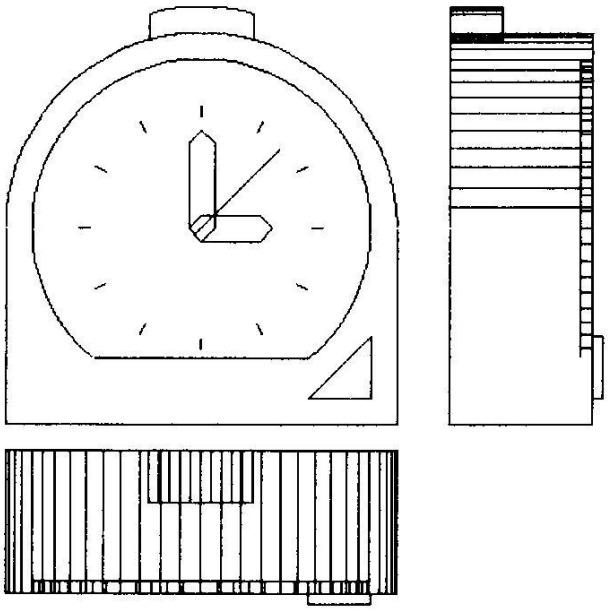
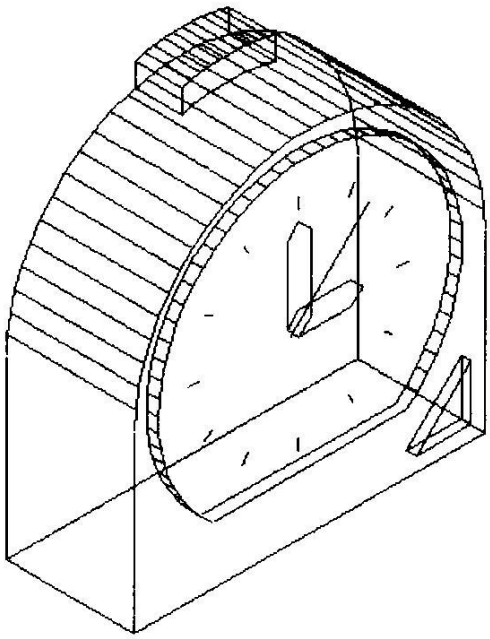


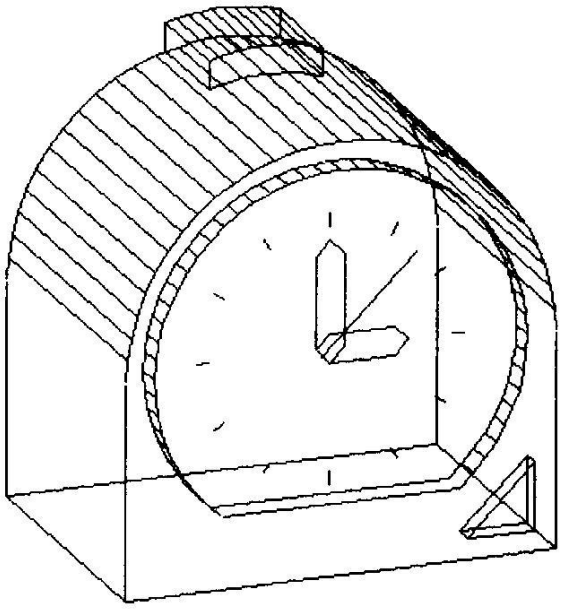
Рисунок 5 - Диметрия и изометрия

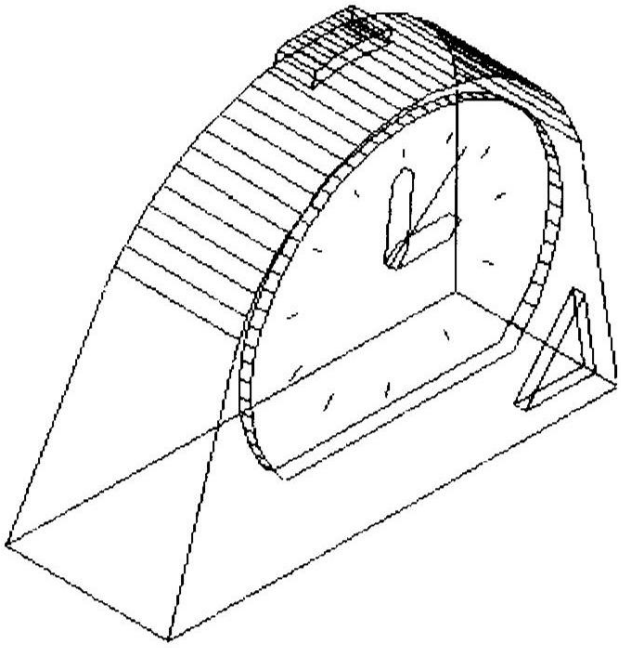
Варианты заданий

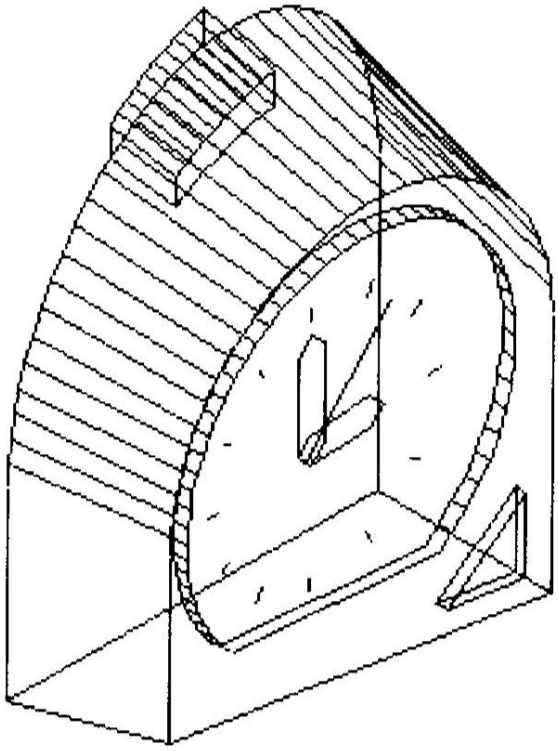
Вариант 1

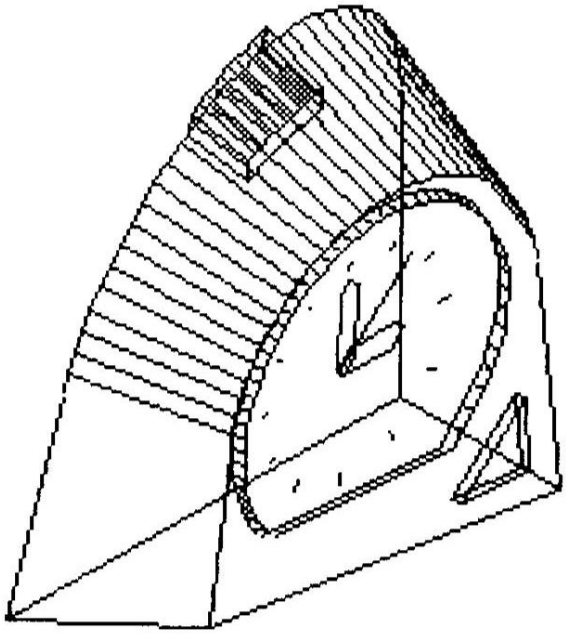


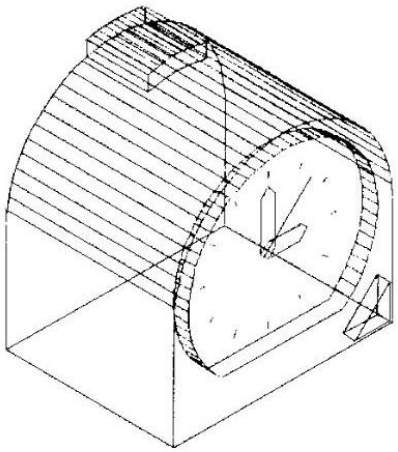


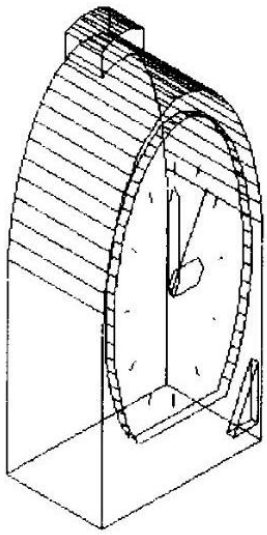


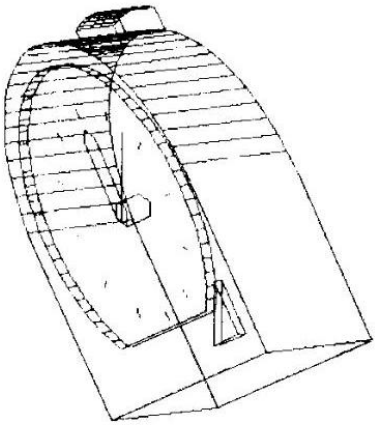


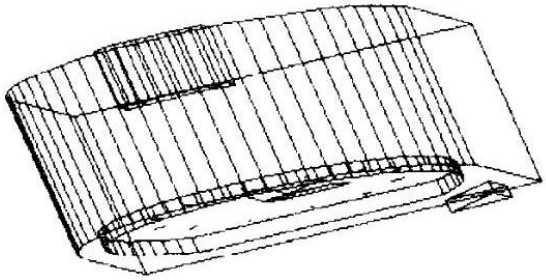


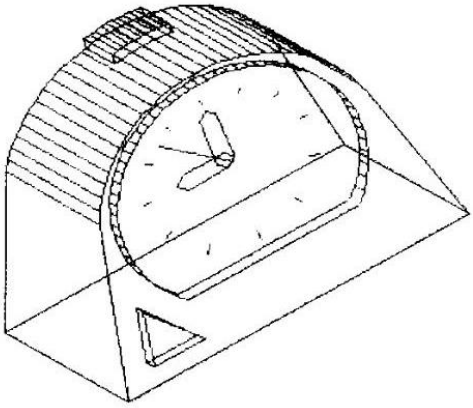


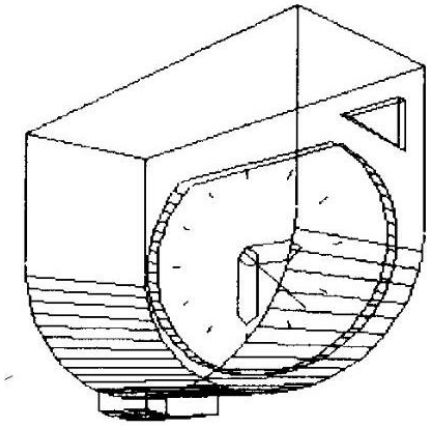




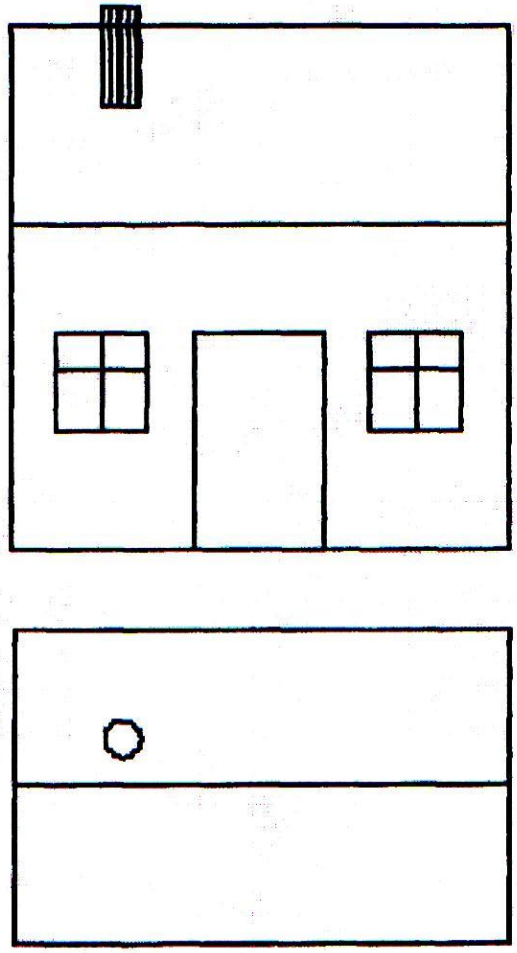


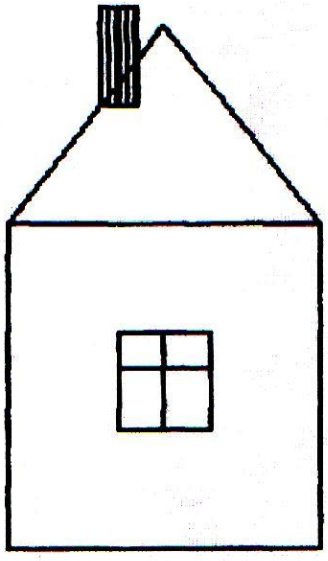


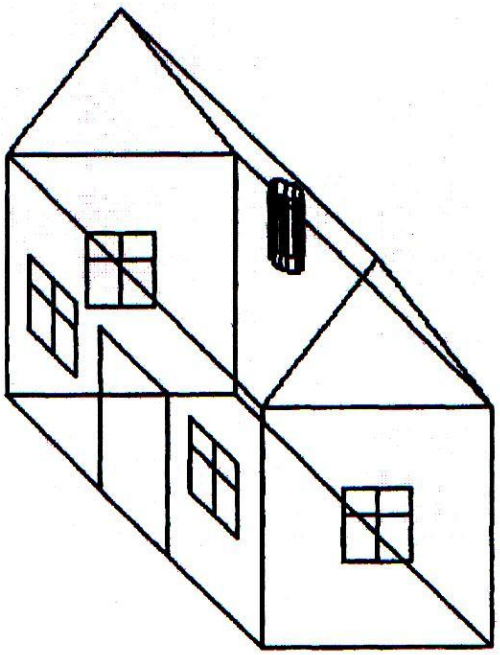


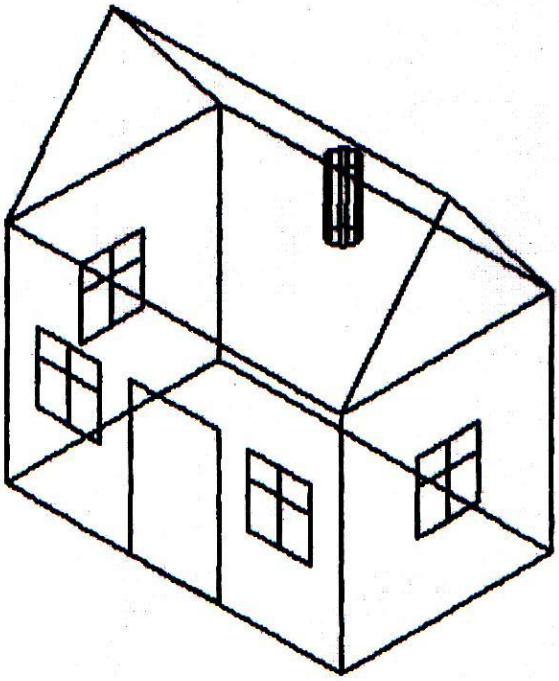


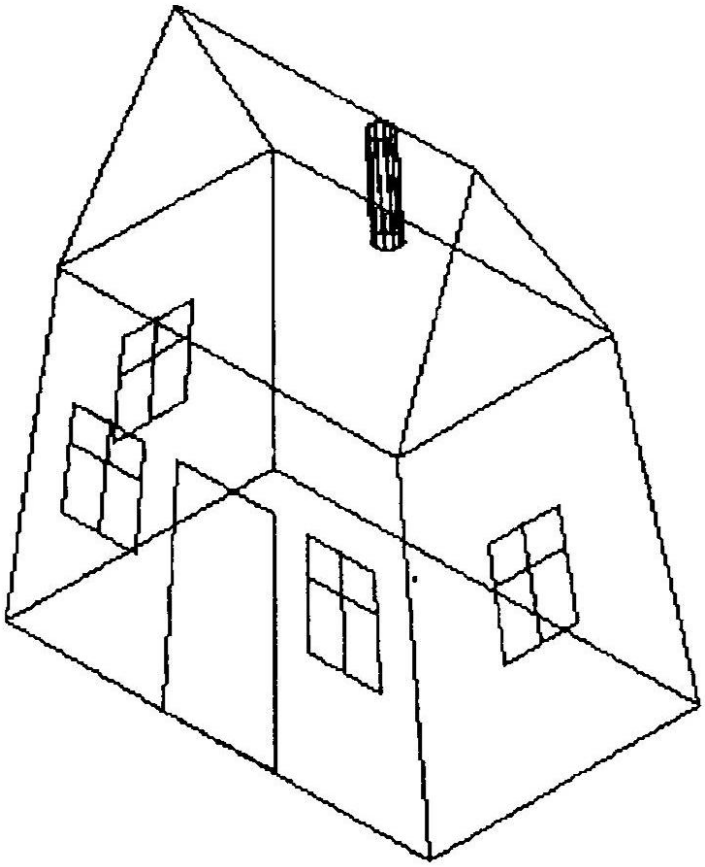
Вариант 2

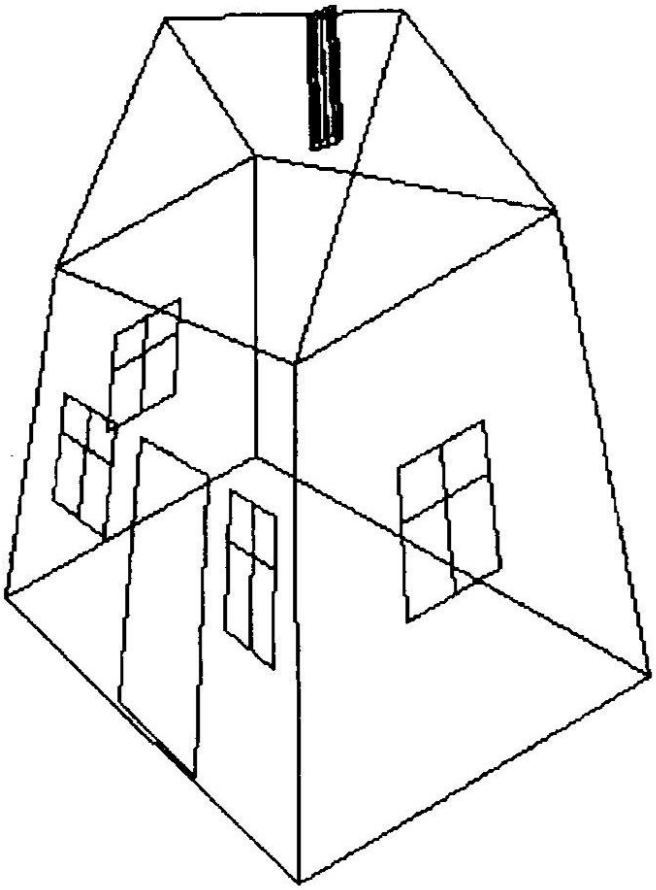




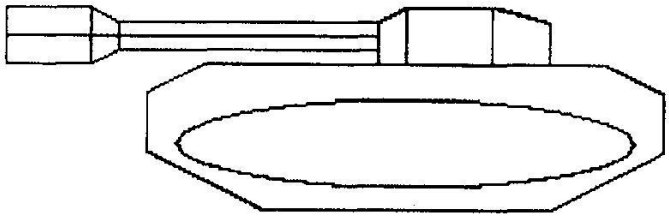


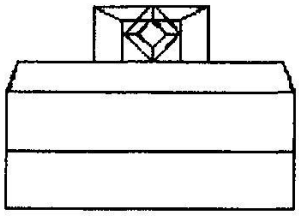


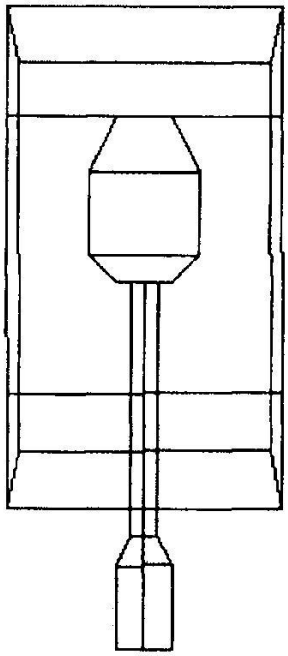


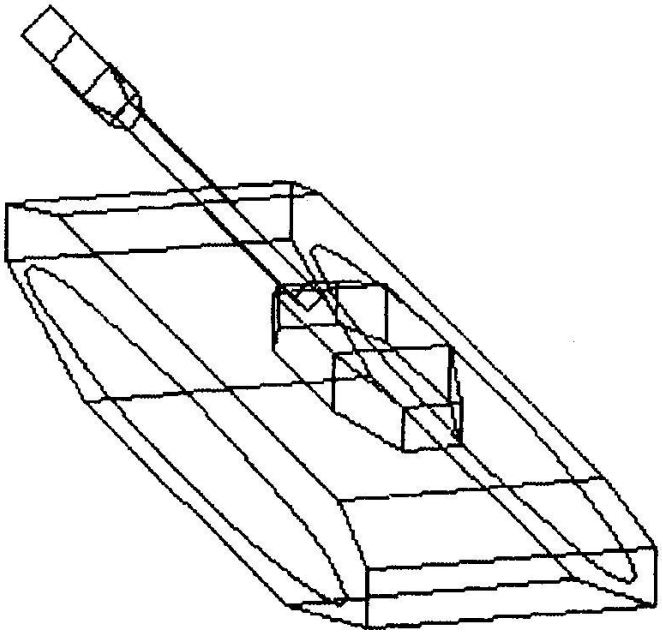


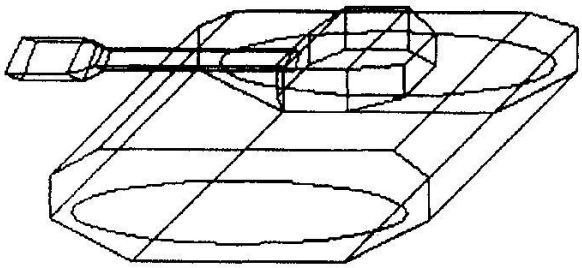
Вариант 3

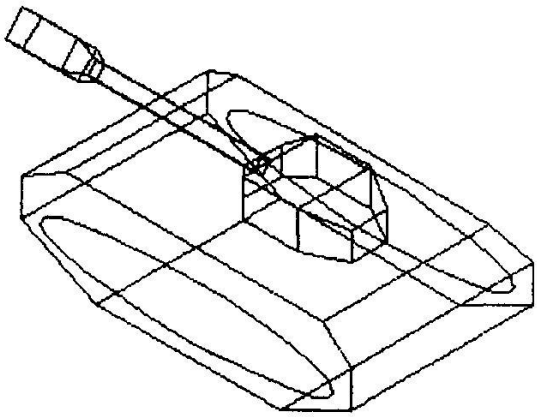


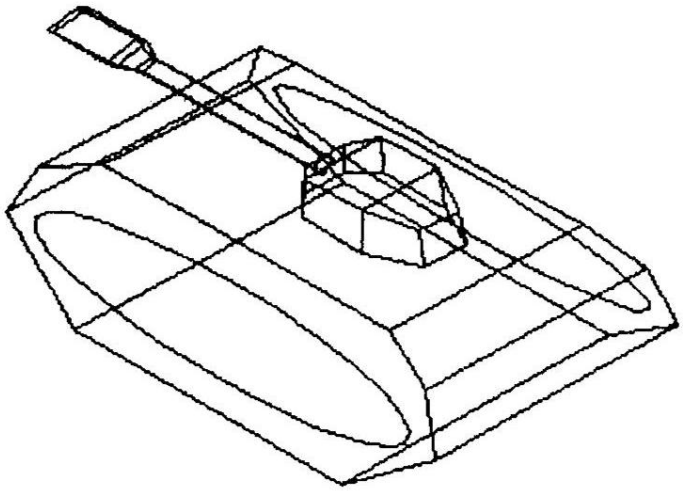


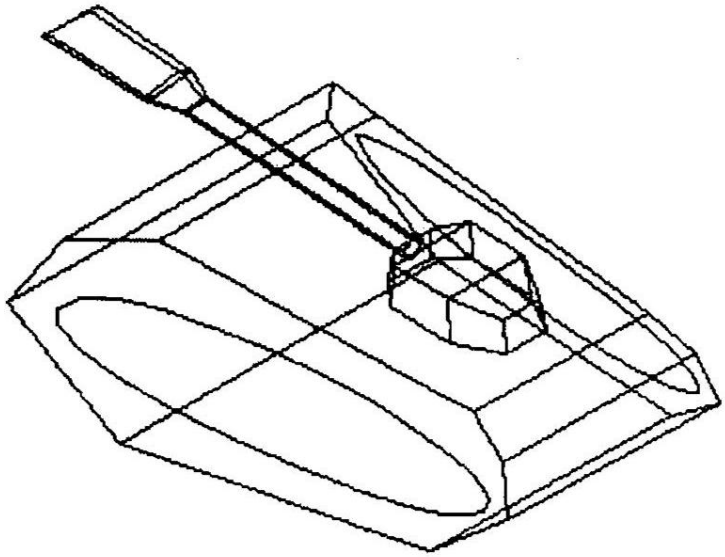


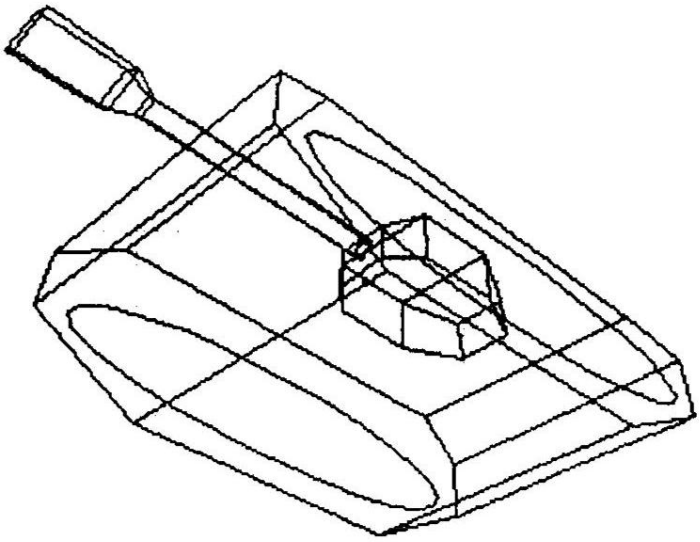




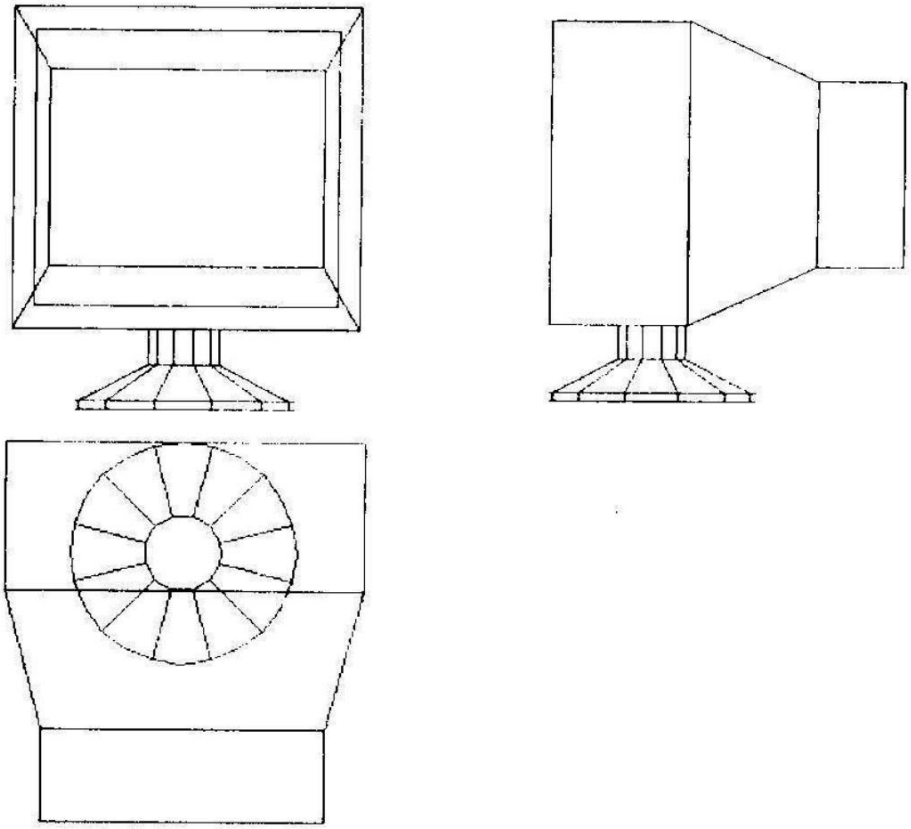


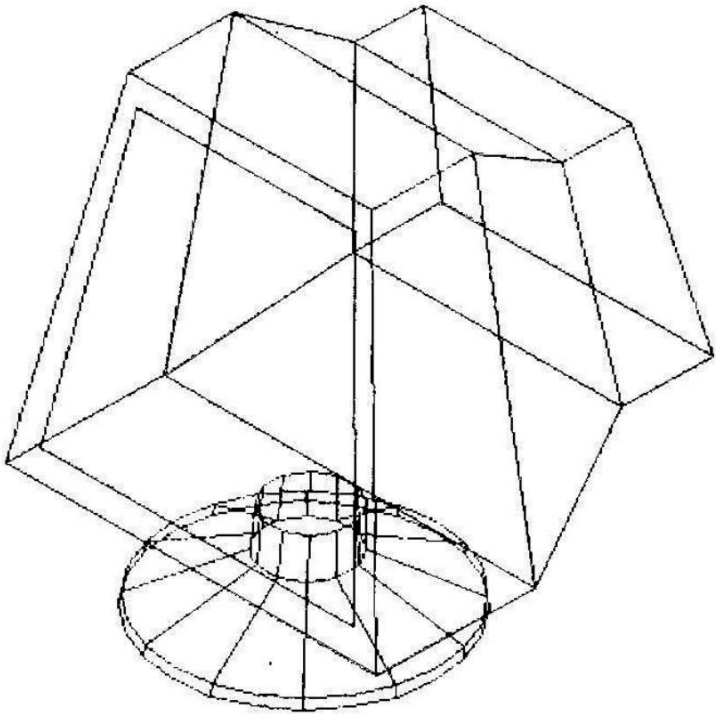


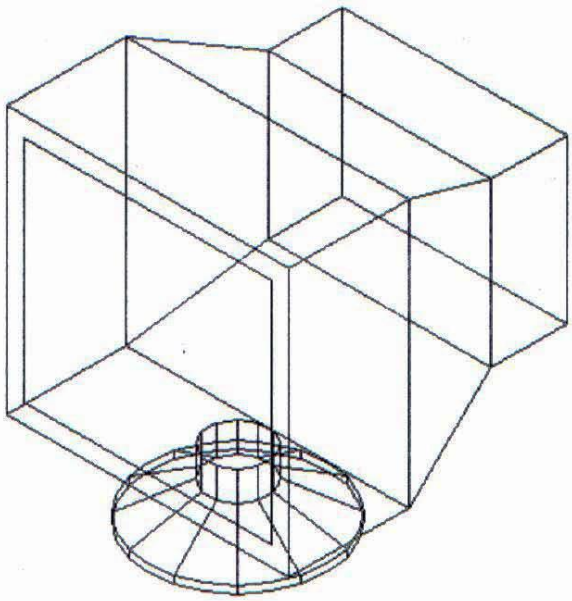


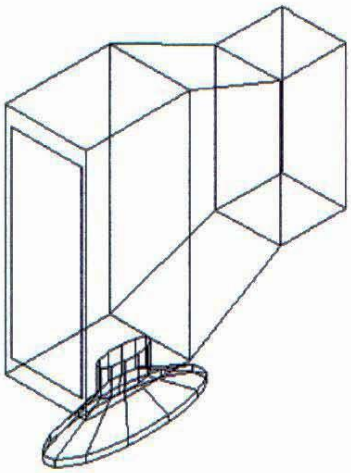


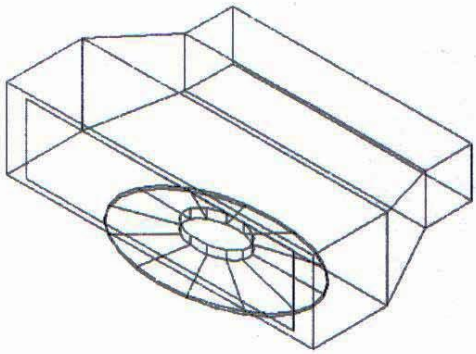
Вариант 4

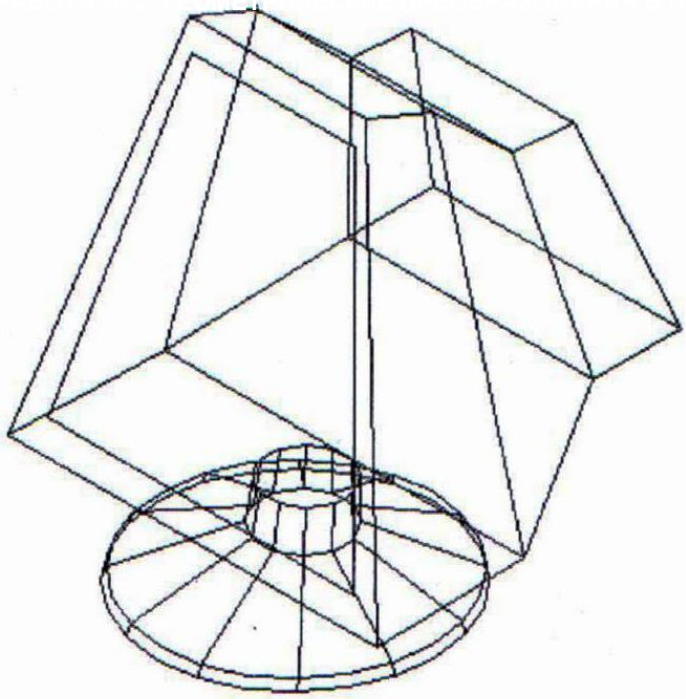


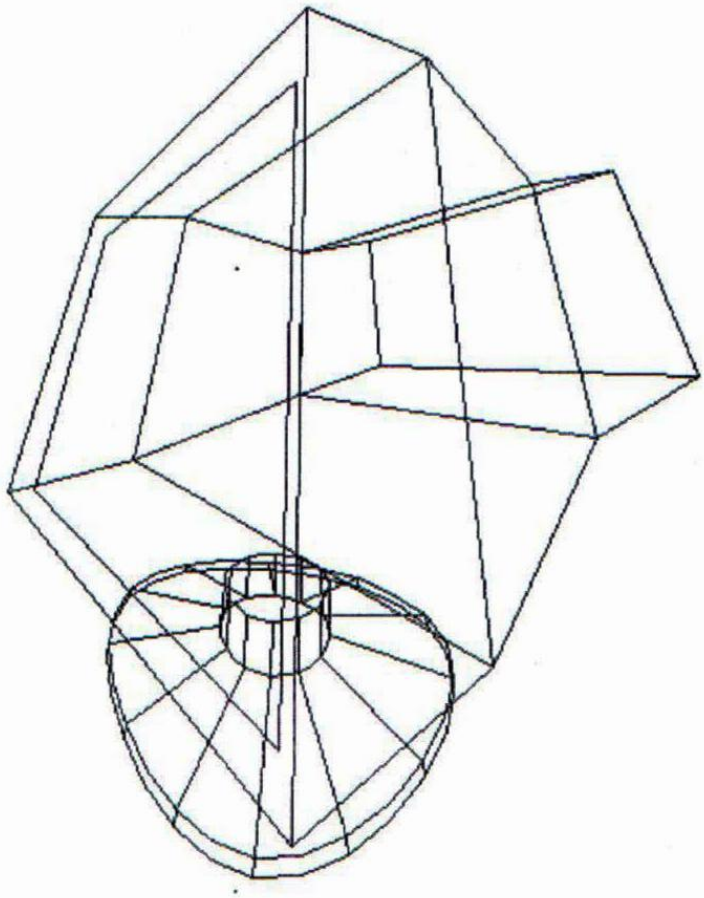




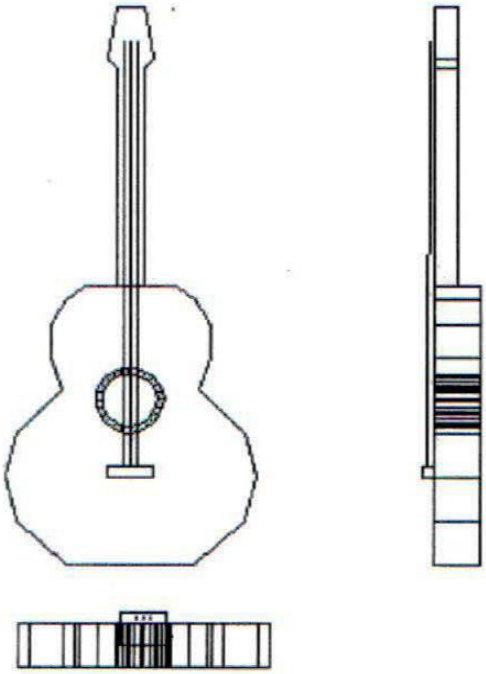


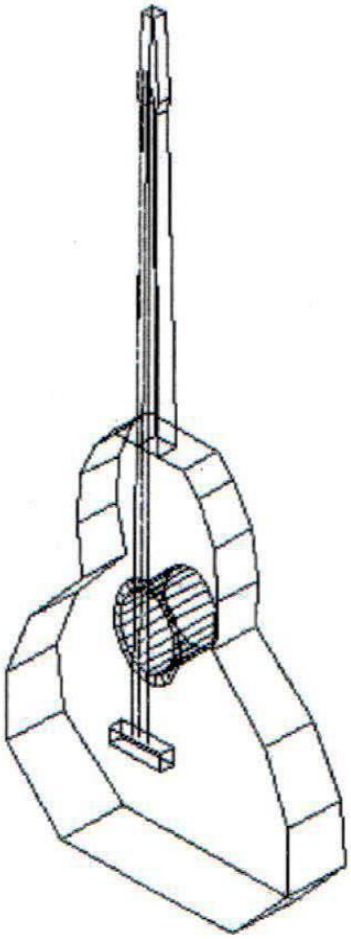


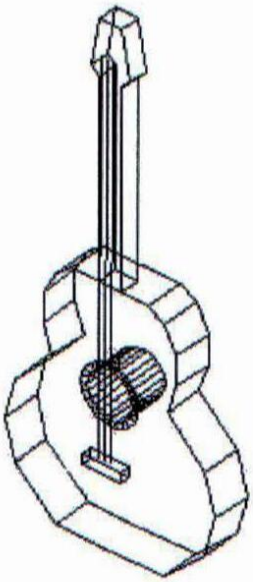


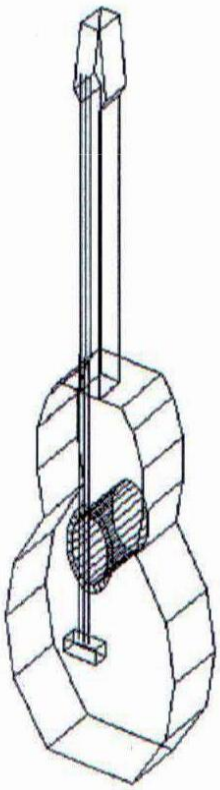


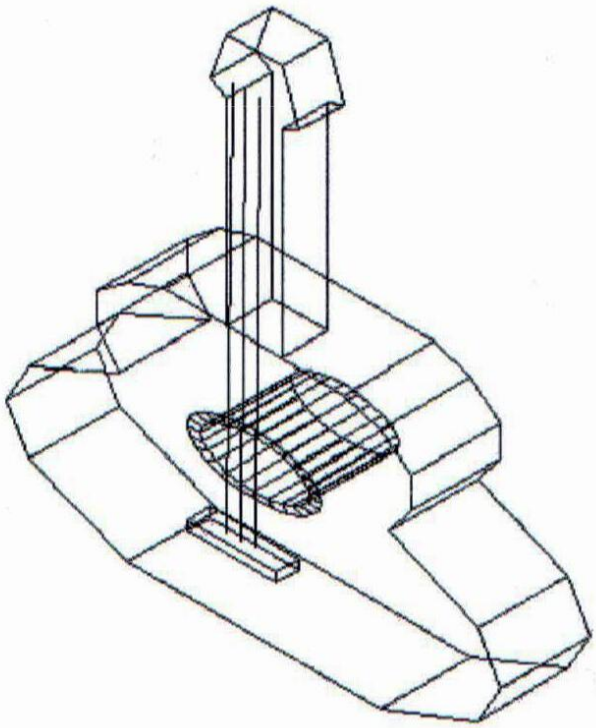
Вариант 5

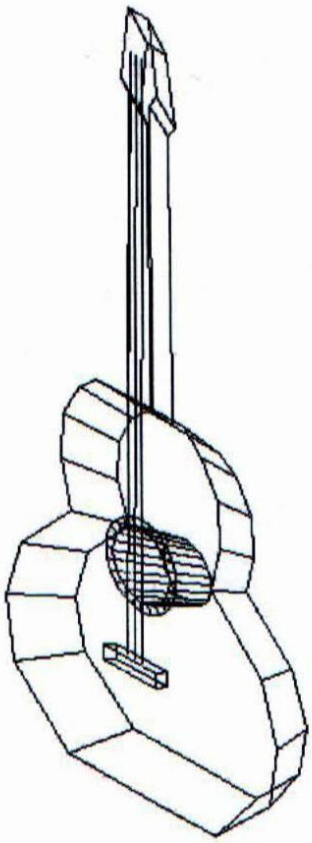


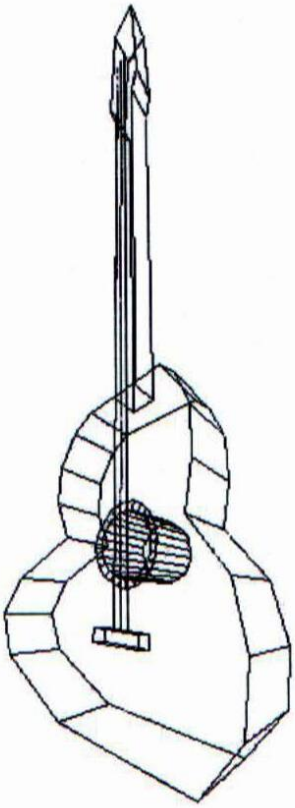




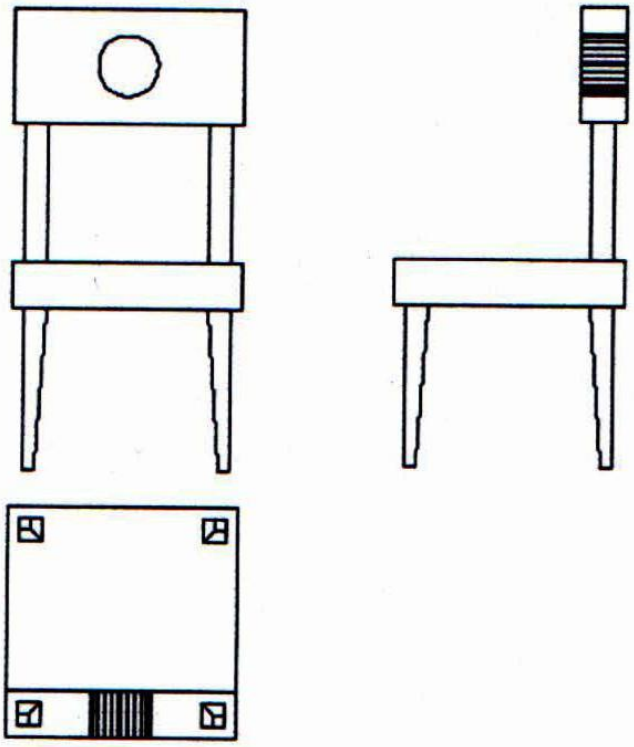


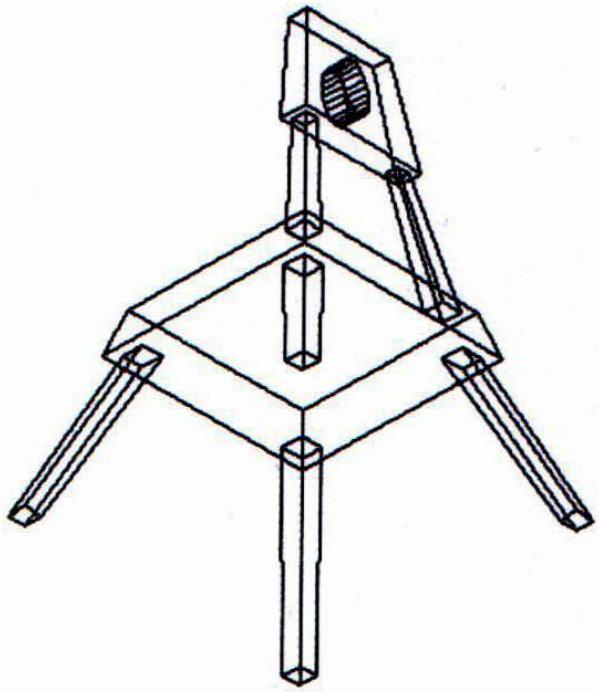


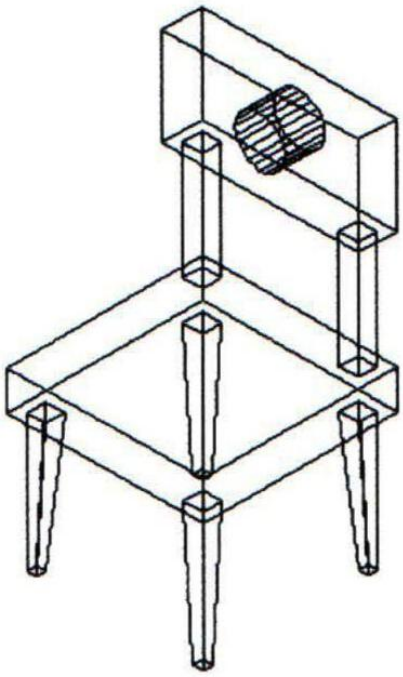


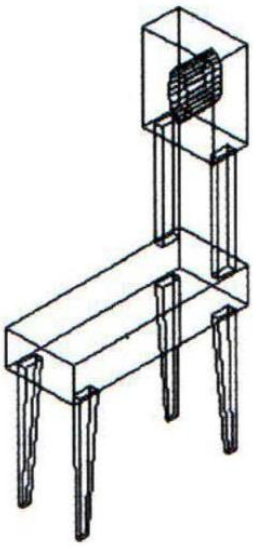


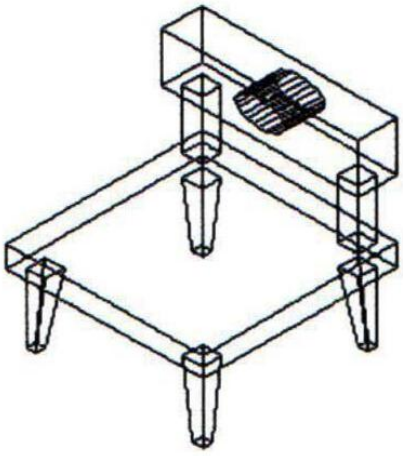
Вариант 6

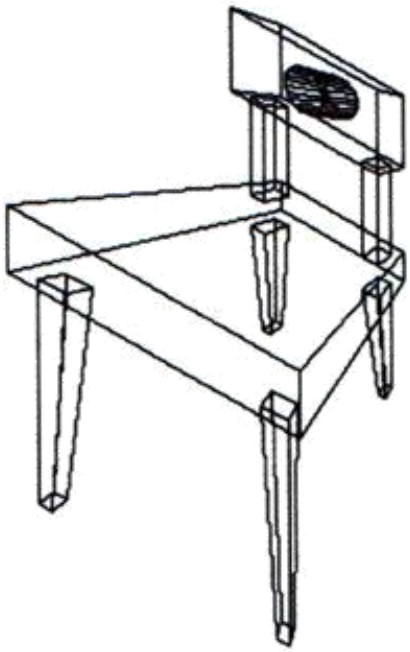


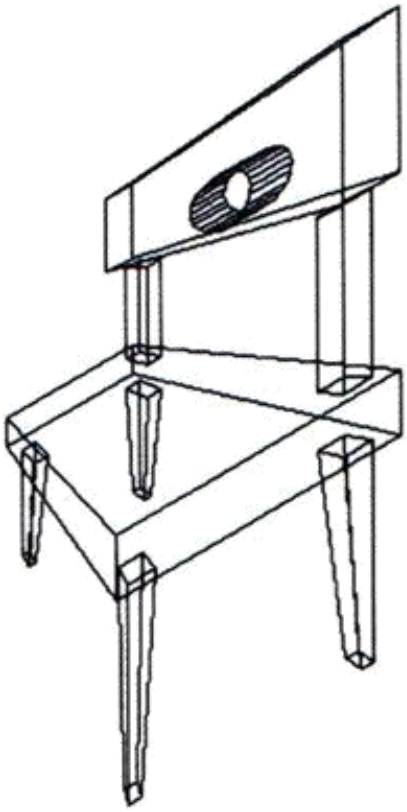












Оформление отчета по лабораторной работе

В отчете должны быть представлены результаты всех этапов лабораторной работы. Структура отчета следующая:

1. Постановка задачи и выбор объекта. Результаты предварительной работы с изображениями объекта – оцифровка объекта.
2. Краткое математическое описание выполняемых геометрических преобразований (единичных преобразований и композиций) в матричной форме.
3. Листинг программы, реализующей геометрические преобразования объекта.
4. Результаты выполненных преобразований в виде копий графического экрана: ортогональные проекции, прямоугольные аксонометрические проекции (диметрия и изометрия) и три вида перспективных проекций (с разным количеством точек схода лучей).

Лабораторная работа 4 Растровые алгоритмы построения контура фигур

Цель работы: освоить алгоритмы о контура фигур

Теоретическая часть

Алгоритм Брезенхема рисования линии

Пусть необходимо построить отрезок начало которого имеет координаты ($x1$, $y1$), а конец ($x2$, $y2$). Обозначим $dx=(x2-x1)$, $dy=(y2-y1)$. Не нарушая общности, будем считать, что начало отрезка совпадает с началом координат, и прямая имеет вид $y=\frac{dy}{dx}x$, где $\frac{dy}{dx}\in[0,1]$, т.е. $dy<dx$. Считаем что начальная точка находится слева. Пусть на (i- 1)-м шаге текущей точкой отрезка является $P_{i-1}=(r, q)$. Необходимо сделать выбор следующей точки S_i или T_i .

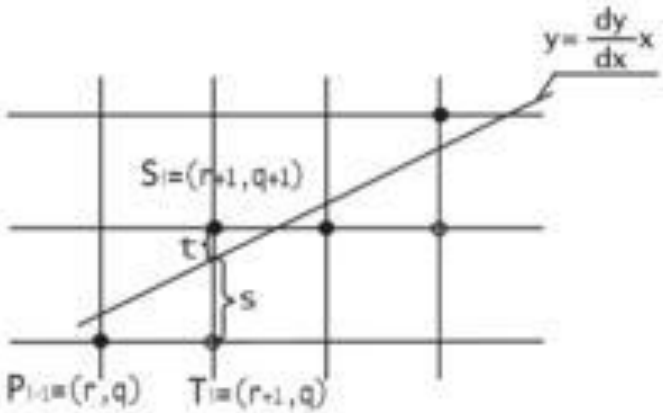
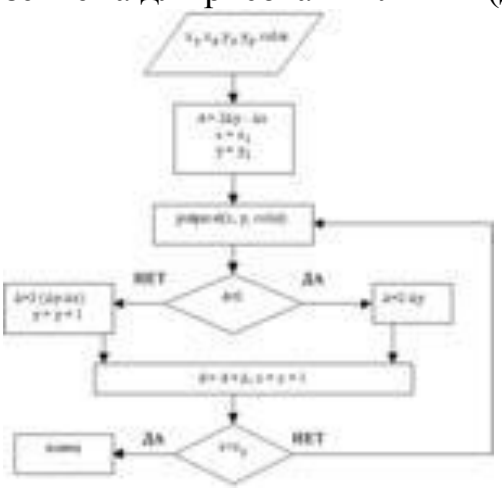


Рисунок 1- Рисование отрезков прямых по методу Брезенхема

Алгоритм:

1. Берем первую точку $x_1 = x_{нач}$.
2. Рассчитываем $\Rightarrow d_1 = 2dy - dx$.
3. Если на предыдущем шаге $d_i < 0$, тогда $d_{i+1} = d_i + 2dy$ и $y_{i+1} = y_i + 1$.
   Если же на предыдущем шаге $d_i \geq 0$, то $d_{i+1} = d_i + 2(dy - dx)$ и $y_{i+1} = y_i$.
4. $x_i = x_i + 1$;
5. Если $x_i > x_{кон}$, то перейти на пункт 3, иначе − конец.

Блок схема алгоритма Брезенхема для рисования линии (для случая когда $dy\ dx$)



Растровая развёртка окружности

Существует несколько очень простых, но не эффективных способов преобразования окружностей в растровую форму. Например, рассмотрим для простоты окружность с центром в начале координат. Ее уравнение записывается как

$$.x^2 + y^2 = R^2. \qquad (1)$$

Решая это уравнение относительно y, получим

$$.y = \pm \sqrt{R^2 \quad x^2} \quad . \qquad (2)$$

Вторым простым методом растровой развертки окружности является использование параметрического представления окружности и вычислений x и y по формулам (3), (4)

$$x = R\ cos\alpha, \qquad (3)$$

$$y = R\ sin\alpha\ . \qquad (4)$$

при пошаговом изменении угла α от 0 до 360 .

Для упрощения алгоритма растровой развёртки стандартной окружности можно воспользоваться её симметрией относительно координатных осей и прямых $y = \pm x$; в случае, когда центр окружности не совпадает с началом координат, эти прямые необходимо сдвинуть параллельно так, чтобы они прошли через центр окружности. Тем самым достаточно построить растровое представление для 1/8 части окружности, а все оставшиеся точки получить симметрией (рис. 2).

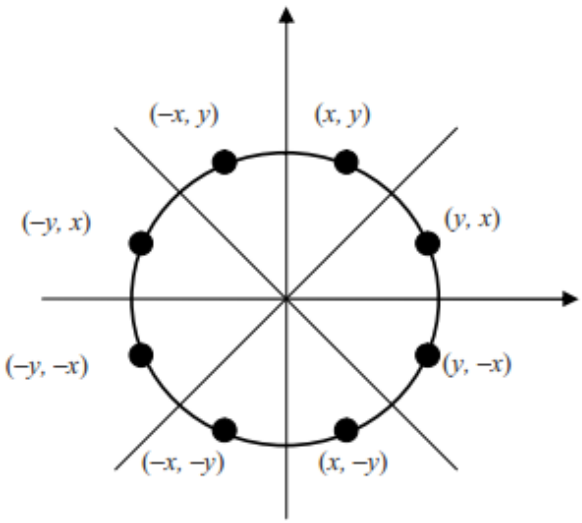


Рисунок 2 - Восьмисторонняя симметрия

Итак, если точка (x, y) лежит на окружности, то легко вычислить семь точек, принадлежащих окружности, симметричных этой. То есть, имея функцию вычисления значения y по $x=0...R/SQRT(2)$ для построения дуги от 0 до 45 .Пример функции для вывода по координатам одной точки сразу восьми точек, симметричных центру окружности:

```
void Circle_Pixel(int x0, int y0, int x, int y,
int color);

{ putpixel(x0 + x, y0 + y, color);
  putpixel(x0 + y, y0 + x, color);
  putpixel(x0 + y, y0 - x, color);
  putpixel(x0 + x, y0 - y, color);
  putpixel(x0 - x, y0 - y, color);
  putpixel(x0 - y, y0 - x, color);
  putpixel(x0 - y, y0 + x, color);
  putpixel(x0 - x, y0 + y, color); }
```

Алгоритм Брезенхейма для окружности

Рассмотрим участок окружности из второго октанта $x \in [0, R/\sqrt{2}]$ (рис. 2). Опишем алгоритм Брезенхейма для этого участка окружности.

На каждом шаге алгоритм выбирает точку $P_i(x_i, y_i)$, которая является ближайшей к истинной окружности. Идея алгоритма заключается в выборе ближайшей точки при помощи управляющих переменных, значения которых можно вычислить в пошаговом режиме с использованием небольшого числа сложений, вычитаний и сдвигов.

Рассмотрим небольшой участок сетки пикселей, а также возможные способы (от A до E) прохождения истинной окружности через сетку (рис. 3).

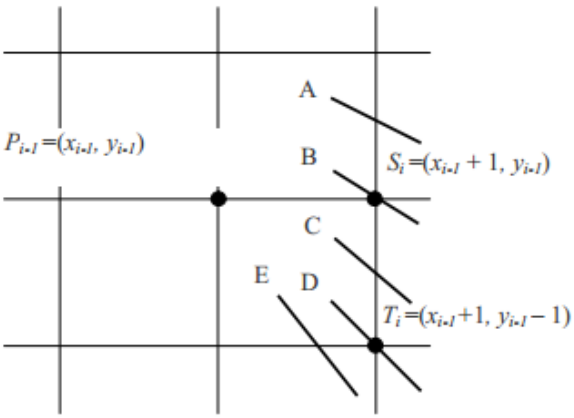


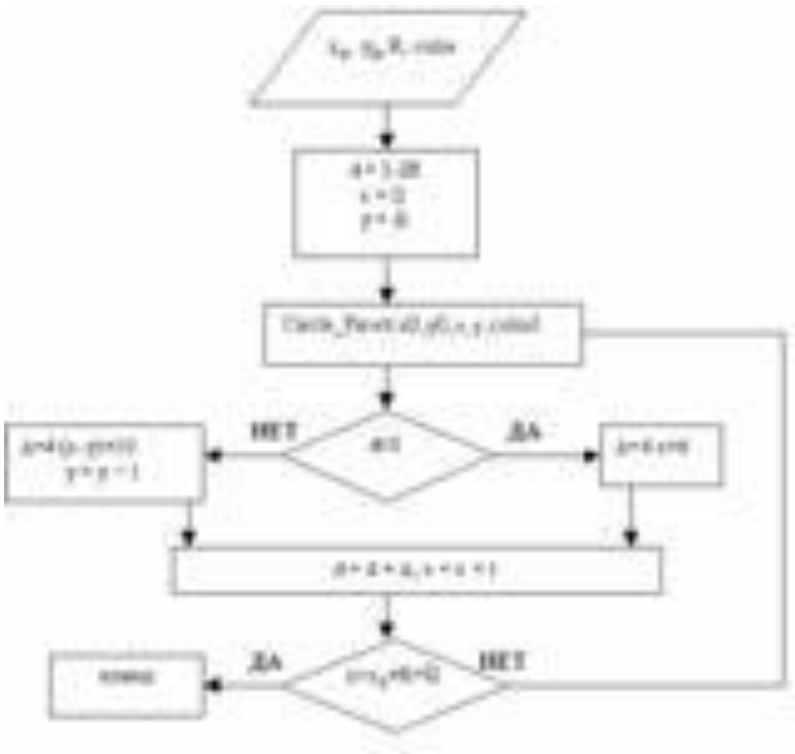
Рисунок 3-Варианты прохождения окружности через растровую сетку

Предположим, что точка P_{i-1} была выбрана как ближайшая к окружности при $x = x_{i-1}$. Далее необходимо найти, какая из точек (S_i или T_i) расположена ближе к окружности при $x = x_{i-1} + 1$.

Алгоритм:

1. Берем первую точку $x_1 = x_{нач}$.
2. Рассчитываем $d_1 = 3 - 2R$.
3. Если $d_i < 0$, то $d_{i+1} = d_i + 4x_{i-1} + 6$. и выбирается S_i. Если $d_i \geq 0$, то $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$ и выбирается T_i.
4. $x_i = x_i + 1$;
5. Если $x_i > x_{кон}$, то перейти на пункт 3, иначе − конец.

Блок-схема алгоритма Брезенхейма для окружности



Алгоритм хорош тем что отсутствуют операции с плавающей точкой, а также операции деления и извлечения корня.

Растровая развёртка эллипса

Простым методом растровой развертки эллипса является использование вычислений x и y по формулам

$$x = Rx \cos\alpha, \quad (5)$$

$$y = Ry \sin\alpha \quad (6)$$

при пошаговом изменении угла α от 0 до 360 .

Можно также использовать симметрию эллипса.

Существует модификация алгоритма Брезенхейма для эллипса.

Кривая Безье

Разработана математиком Пьером Безье. Кривые и поверхности Безье были использованы в 60-х годах компанией «Рено» для компьютерного проектирования формы кузовов автомобилей. В настоящее время они широко используются в компьютерной графике.

Кривые Безье описываются в параметрической форме:

$$x = P_x(t), \quad y = P_y(t). \tag{7}$$

Значение *t* выступает как параметр, которому соответствуют координаты отдельной точки линии. Параметрическая форма описания может быть удобнее для некоторых кривых, чем задание в виде функции *y=?(x)*, поскольку функция *?(x)* может быть намного сложнее, чем *Px(t)* и *Py(t)*, кроме того, *?(x)* может быть неоднозначной.

Многочлены Безье для *Px* и *Py* имеют такой вид:

$$P_x(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} x_i, \quad P_y(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} y_i, \tag{8}$$

где x_i и y_i — координаты точек-ориентиров P_i, а величины C_m^i — это известные из комбинаторики, так называемые сочетания (они также известны как коэффициенты бинома Ньютона):

$$C_m^i = \frac{m!}{i!(m-i)!}. \tag{9}$$

Значение m можно рассматривать и как степень полинома, и как значение, которое на единицу меньше количества точек-ориентиров.

Рассмотрим кривые Безье, классифицируя их по значениям *m*.

1. <u>*m* = 1 (по двум точкам)</u>

Кривая вырождается в отрезок прямой линии, которая определяется конечными точками P_0 и P_1, как показано на рисунке 5:

$$P(t) = (1-t)P_0 + tP_1 \tag{10}$$

2. <u>*m* =2 (по трем точкам, рисунке 5:</u>

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2. \tag{11}$$

3. <u>*m* = 3 (по четырем точкам, кубическая, рис. 6)</u>. Используется довольно часто, в особенности в сплайновых кривых:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3. \quad (12)$$

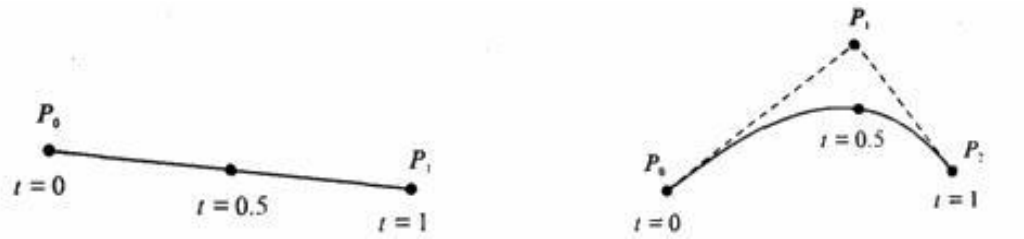


Рисунок 5-Кривая Безье (m=1) и кривая Безье (m=2)

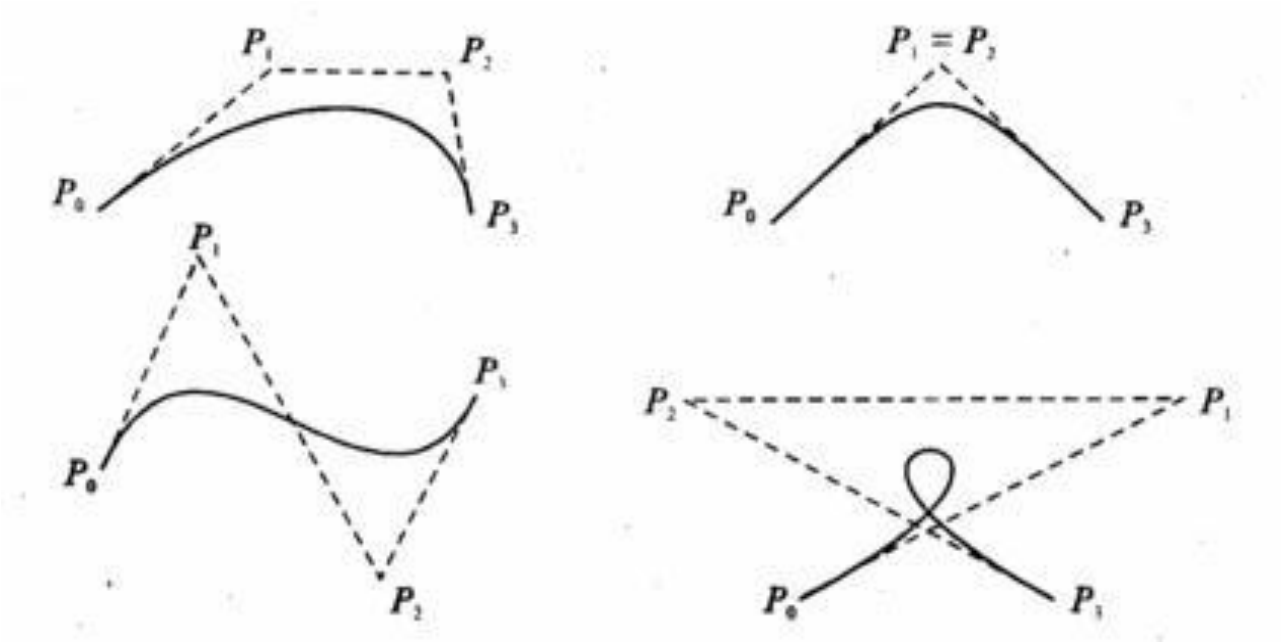


Рисунок 6 - Кубические кривые Безье (m=3)

Геометрический алгоритм для кривой Безье

Этот алгоритм позволяет вычислить координаты (x, y) точки кривой Безье по значению параметра t.

1. Каждая сторона контура многоугольника, который проходит по точкам-ориентирам, делится пропорционально значению t.

2. Точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на единицу меньше, чем количество узлов предшествующего контура.

3. Стороны нового контура снова делятся пропорционально значению t. И так далее. Это продолжается до тех пор, пока не будет получена единственная точка деления. Эта точка и будет точкой кривой Безье (рис.7).

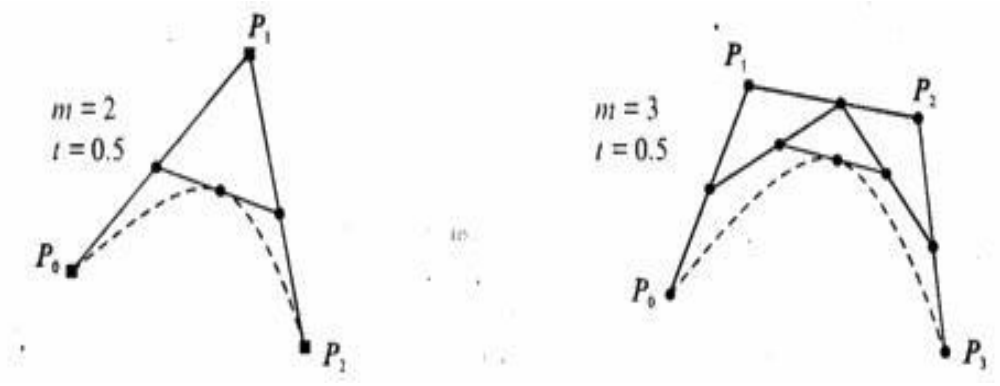


Рисунок 7- Геометрический алгоритм для кривых Безье

Пример на языке C++:

```
int m=3; // степень полинома
int n=6; // количество точек на кривой float t;
for (int l=0; l<=n; l++)
{
 t=(1.0/n)*l;
 for (int i=0; i<2; i++)
        for (int j=0; j<=m; j++)
               R[i][j]=(P[i][j]); // вспомогательный
// массив дублирует координаты точек – ориентиров.
  for (int i=0; i<2; i++) // координаты x и y
   for (int j=m; j>=0; j--)
    for (int k=0; k<j; k++)
        { R[i][k]+= t*(R[i][k+1]-R[i][k]);
          T[i][l]=R[i][k] ;//координаты точек на кривой

        }

 }
```

Уравнения кривой Безье можно записать в матричном виде:

$$P(t) = [T][N][G] \tag{13}$$

где для m = 3:

$$P(t) = [T][N][G] = [t^3 \quad t^2 \quad t \quad 1]\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{14}$$

Матрица N представляет собой коэффициенты при соответствующих t (Первый столбец при P_0, второй при P_1, третий при P_2 и четвертый при P_3. Первая строка при t^3, вторая при t^2, третья при t и четвертая свободный член). P_0, P_1, P_2 и P_3 вершины многоугольника Безье.

Аналогично для $m = 4$:

$$P(t) = [t_4 \quad t_3 \quad t_2 \quad t \quad 1] \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \tag{15}$$

B-сплайны

Кривая, представленная в виде кубического B-сплайна, в общем случае может проходить через любые управляющие точки, однако она непрерывна и, кроме того, непрерывностью изменения обладают ее касательный вектор и кривизна (т. е. первая и вторая производные кривой непрерывны в конечных точках) в отличие от формы Безье, у которой в конечных точках непрерывны лишь первые производные (но которые проходят через управляющие точки). Таким образом, можно утверждать, что форма B-сплайнов «более гладкая», чем другие формы. B-сплайн описывается следующей формулой:

$$x(t) = TM_s G_{i-1} \tag{16}$$

где

$$M_s = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \tag{17}$$

При аппроксимации управляющих точек P_1, P_2, …, P_n последовательностью B-сплайнов необходимо применять между каждой парой соседних точек геометрические матрицы. Для аппроксимации в интервале, близком к точкам P_i и P_{i+1}, используется

$$G_i = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}, \quad 2 \le i \le n-2 \tag{18}$$

Пример 1. Построение дерева (рис. 8)

Точки ориентиры для сегментов кривой Безье, сегменты нумеруются по часовой стрелке:

1) P=	4	0
	2	1
	2	3
	3.5	2.7
2) P=	3.5	2.7
	3	4.5
	3	7
	3.7	5
3) P=	3.7	5

4) P= 4 8
4 8
5 5
5 5
6 7
6 5
5 3

5) P= 5 3
6 3.5
6 1.5
5 0

Пример 2. Построение цветка (рис.9).

Точки ориентиры для сегментов кривой Безье:

1) P= 3 2
2.5 2.5
2.5 1.5
3 2

2) P= 3 2
3.5 2.5
3.5 1.5
3 2

3) P= 3 2
2.5 2.5
3.5 2.5
3 2

4) P= 3 2
2.5 1.5
3.5 1.5
3 2

Стебель:

5) P=3 0.75
3.5 1.5
2 1
3 2

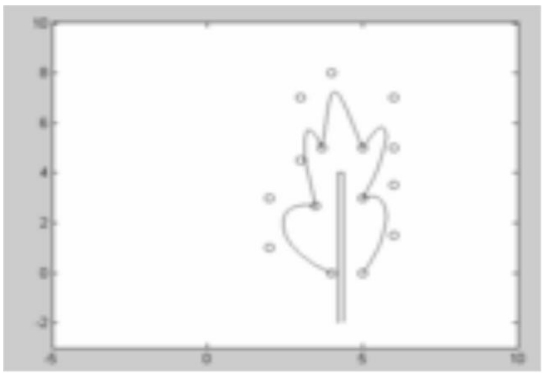


Рисунок 8- Пример рисунка – дерево

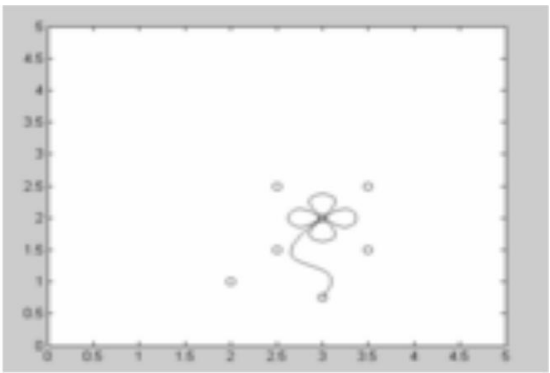


Рисунок 9- Пример рисунка – цветок

Задание Написать и отладить программу на языке C/C++ для вывода рисунка по варианту (приложение). Для этого:

Написать необходимые функции для реализации растровой развертки графических примитивов: отрезок (алгоритмом Брезенхема); окружность (алгоритмом Брезенхема); эллипс (параметрический); сплайновые кривые (Безье и / или B-сплайны).

Нарисовать заданный по варианту рисунок (приложение).

Содержание отчета

Титульный лист.

Задание.

Описание методов построения примитивов. Для сплайновых кривых показать точки ориентиры и сегменты.

Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

ПРИЛОЖЕНИЕ

Графические фигуры

Таблица - Варианты заданий

№ вар.	Рисунок	№ вар.	Рисунок
1		2	
3		4	
5		6	

7		8	
9		10	
11		12	
13		14	

15		16	
17		18	
19		20	
21		22	

Лабораторная работа 5 Алгоритмы отсечения фигур

Цель работы: освоить алгоритмы отсечения фигур в программировании

Теоретическая часть

Этап отсечения необходим и присутствует в большинстве задач компьютерной графики для определения реальных областей фигур, которые будут выведены на экран. Это необходимо, если фигура больше или выходит за пределы экрана или окна вывода. Отсечение чаше всего проводится для отбрасывания частей фигуры, выходящих за границу области, которая определяет экран или видимую область некоторого прямоугольного окна, хотя отсечение может проводиться и относительно произвольного многоугольника. Для отсечения могут использоваться как методы отсечения отрезка, так и методы отсечения многоугольников, которые, по сути, являются их модификацией. Если необходимо отсечь более сложную фигуру, то выполняется аппроксимация контура отрезками прямых. При отсечении фигуры порождается новый многоугольник или несколько новых многоугольников.

Рассмотрим эту задачу сначало применительно к отрезкам прямых. Некоторые из них полностью лежат внутри области экрана, другие целиком вне ее, а некоторые пересекают границу экрана. Правильное отображение отрезков означает нахождение точек пересечения их с границей экрана и рисование только тех их частей, которые попадают на экран.

Алгоритм Коэна-Сазерленда

Рассмотрим алгоритм Коэна-Сазерленда для отсечения отрезков прямых. Этот алгоритм позволяет легко определять нахождение отрезка полностью внутри или полностью снаружи окна, и если так, то его можно рисовать или не рисовать, не заботясь об отсечении по границе окна.

Для работы алгоритма вся плоскость в которой лежит окно разбивается на девять подобластей или квадратов, как показано на рисунке 1.

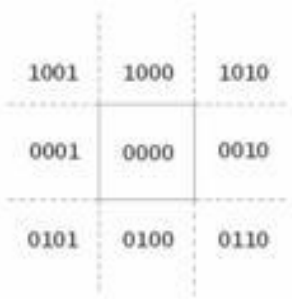


Рисунок 1 - Разбиение на подобласти в методе Коэна-Сазерленда

Окну соответствует область обозначенная кодом 0000. Конечным точкам отрезка приписывается 4-битный код «вне/внутри» в зависимости от

нахождения отрезка в соответствующей подобласти. Каждому биту присваивается значение 1 в соответствии со следующим правилом.

Бит 1 точка находится выше окна;
Бит 2 – точка находится ниже окна;
Бит 3 точка находится справа от окна;
Бит 4 точка находится слева от окна;

Иначе биту присваивается нулевое значение. Значения этих битов для конечных точек отрезков легко определить по знакам соответствующих разностей: $(y_{max}-y)$ - для 1-го бита, $(y-y_{min})$ - для 2-го бита, $(x_{max}-x)$ - для 3-го бита и $(x-x_{min})$ - для 4-го бита.

Для каждого отрезка рассчитываются коды концов ($К_1$, $К_2$) затем производится экспресс анализ:

- если $^K1 \wedge ^K2 \neq 0$, тогда отрезок лежит вне поля вывода – отрезок отбрасывается;
- если $^K1 = ^K2 = 0$, тогда отрезок полностью лежит внутри поля вывода – отсечение не нужно, отрезок полностью прорисовывается;
- если $^K1 \wedge ^K2 = 0$, отрезок может частично лежать внутри поля вывода – необходимо отсечение по полю вывода.

В этом случае применяется последовательное разделение отрезка, так что на каждом шаге конечная точка отрезка с ненулевым кодом вне/внутри заменяется на точку, лежащую на стороне окна или на прямой содержащей сторону. При этом порядок перебора сторон окна не имеет значения.

Когда $^K1 \wedge ^K2 = 0$ необходимо отсекать отрезок по границам поля вывода, отсечение происходит последовательно по всем сторонам рисунок 2.

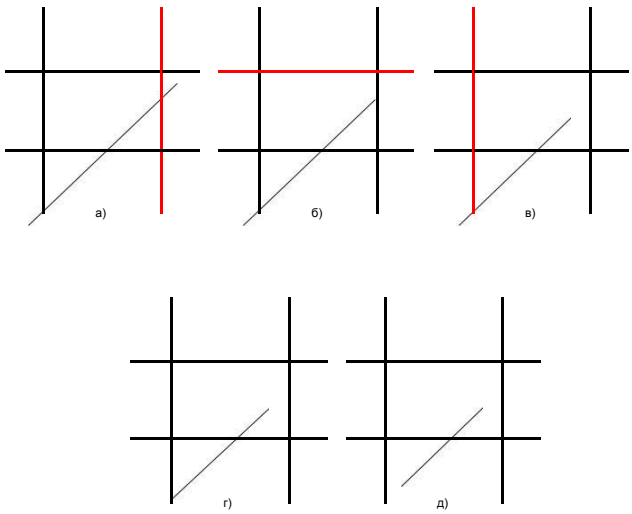


Рисунок 2 - Отсечение отрезка по прямоугольной области

На рисунке 2 жирным выделено ребро по которому происходит отсечение. Также надо отметить, что точки лежащие на границе поля вывода принадлежат полю вывода.

На каждом шаге отсечения вычисляются новые координаты одной точки.

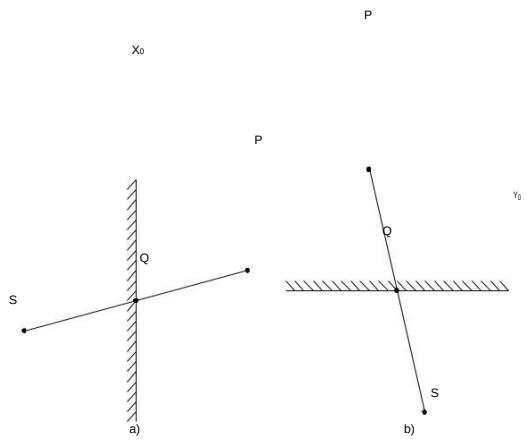


Рисунок 3 - Варианты пересечения отрезка и стороны окна.

Для нахождения координат точки $Q(x_Q, y_Q, z_Q)$ на рисунке 3 а) используется формула (1); для точки $Q(x_Q, y_Q, z_Q)$ рисунке 3 б) − формула (2).

$$y_Q = y_S + \frac{x_Q - x_S}{x_P - x_S}(y_P - y_S) \tag{1}$$

$$x_Q = x_S + \frac{y_Q - y_S}{y_P - y_S}(x_P - x_S) \tag{2}$$

Алгоритм Сазерленда-Ходгмана

Теперь рассмотрим операцию отсечения многоугольника. Решение общей задачи сводится к решению ряда простых и похожих подзадач. Примером такой подзадачи является отсечение многоугольника относительно одной отсекающей границы. Последовательное решение четырех таких задач позволяет провести отсечение относительно прямоугольной области (рис. 4).

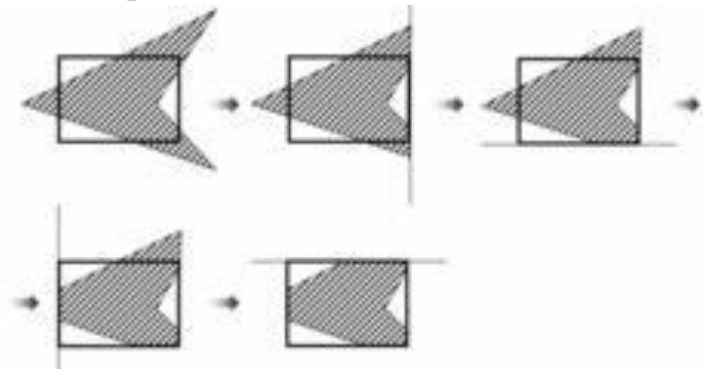


Рисунок 4 - Последовательное отсечение многоугольника

На вход алгоритма поступает последовательность вершин многоугольника V_1, V_2, …, V_n. Ребра многоугольника проходят от V_i к V_{i+1} от V_n к V_1. С помощью алгоритма производится отсечение относительно ребра и выводится другая последовательность вершин, описывающая усеченный многоугольник.

Алгоритм «обходит» вокруг многоугольника от V_n к V_1 и обратно к V_n, проверяя на каждом шаге соотношение между последовательными вершинами и отсекающей границей. Необходимо проанализировать четыре случая рисунке 5.

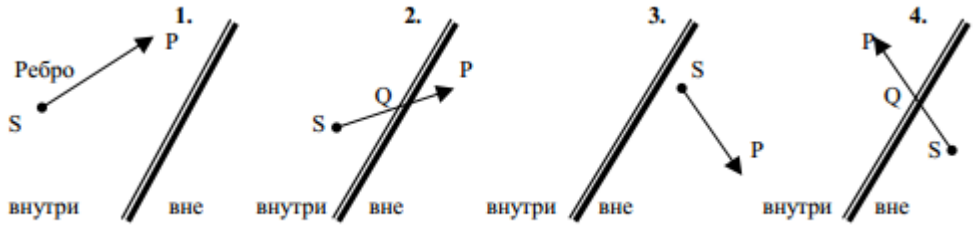


Рисунок 5 - Случаи, возникающие при отсечении многоугольников

Может также возникнуть вариант, когда ребро совпадает с границей.

Рассмотрим случаи на рисунке 5:

1. Добавить в список рисуемых вершин P;
2. Добавить в список Q.
3. Ничего не добавляется.
4. Добавить в список P,Q.

При отсечении многоугольника описанным алгоритмом возникает проблема, связанная с тем, что появляются ребра, частично совпадающие с границей окон. Лишние ребра можно устранить, введя дополнительную обработку или воспользовавшись более общим и более сложным алгоритмом Вейлера – Азертона, который позволяет выполнять отсечение произвольного многоугольника по произвольному многоугольнику.

Задание

Написать и отладить программу на языке C/C++ для вывода объекта по полю вывода (приложение). Для этого:

1. Написать необходимые функции для отсечения многоугольника или отрезка по полю вывода.
2. Выполнить задания согласно варианту (приложении):

- нарисовать многоугольник или отрезок;
- показать зону видимости;

- перемещать многоугольник или отрезок по экрану, при прохождении через зону видимости выделять видимую часть другим цветом.

Содержание отчета

Титульный лист.
Задание.
Описание метода отсечения.
Текст программы.

При защите лабораторной работы тестирование программы **обязательно**

ПРИЛОЖЕНИЕ

Варианты заданий

№	Фигура	Движение	Цвет	Поле вывода	Цвет в зоне видимости
1	A(10,40), B(15,15), C(30,25), D(50,18), E(45,35).	диагональ → влево	красный	(100,100, 300, 300)	белый
2	A(10,40), B(15,15).	вниз → диагональ	серый	(200,200, 400, 400)	красный
3	A(10,50), B(15,25), C(30,35), D(50,28), E(45,45).	диагональ до центра экрана →влево	зеленый	(100,100, 300, 300)	красный
4	C(30,35), D(50,28).	диагональ до центра экрана →вниз	желтый	(200,200, 400, 400)	красный
5	A(5,40), C(15,10), B(10,15), D(45,18), E(40,35).	диагональ → вверх	красный	(100,100, 400, 400)	серый
6	D(45,18), E(40,35).	диагональ → влево	синий	(100,100, 300, 300)	зеленый
7	A(15,40), B(20,15), C(35,25), D(55,18), E(50,35).	диагональ до центра экрана →вниз	белый	(100,100, 400, 400)	синий
8	A(15,40), E(50,35).	диагональ →вверх до середины экрана	желтый	(200,200, 400, 400)	красный
9	A(15,45), B(20,20), C(35,30), D(55,25), E(50,40).	вниз → диагональ	серый	(100,100, 300, 300)	красный
10	B(20,20), D(55,25),	диагональ до центра экрана →вверх	желтый	(200,200, 400, 400)	красный
11	A(20,40), B(25,15), C(45,25), D(60,18), E(55,35).	вправо→ диагональ	синий	(100,100, 300, 300)	зеленый

12	D(20,18), E(55,35).	диагональ до центра экрана →вниз	красный	(100,100, 400, 400)	серый
13	A(10,40), B(15,15), C(30,5), D(50,18), E(45,35).	вниз до середины экрана → вправо	серый	(100,100, 300, 300)	красный
14	D(50,18), E(45,35).	вниз → диагональ	синий	(100,100, 400, 400)	зеленый
15	A(20,49), C(40,35) B(25,5), , D(60,40), E(70,65).	диагональ →вверх до середины экрана	желтый	(200,200, 400, 400)	красный
16	A(10,45), E(45,40).	диагональ до центра экрана →вниз	серый	(100,100, 300, 300)	красный
17	A(20,49), C(40,35) B(25,5), , D(60,40), E(70,65).	диагональ до центра экрана →вправо	синий	(200,200, 400, 400)	зеленый
18	A(12,40), B(18,15).	диагональ → влево	серый	(100,100, 400, 400)	красный
19	A(12,40), B(18,15), C(35,25), D(54,18), E(47,35).	диагональ до центра экрана →вверх	зеленый	(100,100, 300, 300)	красный
20	D(54,18), E(47,35).	вниз → диагональ	синий	(200,200, 400, 400)	зеленый

Лабораторная работа 6 Растровые алгоритмы заполнения фигур

Цель работы: освоить растровые алгоритмы заполнения фигур в программировании

Теоретическая часть

Алгоритмы заполнения (закраски) областей делятся на группы: растровой развертки и затравочного заполнения.

В методах растровой развертки область представляет собой многоугольник с заданными координатами концов, ограничивающих его отрезков. Этими методами находится пересечение строк с многоугольником, определяются пары точек, ограничивающих закрашиваемый отрезок, и на основе этого осуществляется закраска.

В методах затравочного заполнения область задается граничными пикселями, которые окрашены в определенный цвет. Кроме того, задается некоторый пиксель внутри области – затравочный. Затем ищутся пиксели, соседние с затравочным, и закрашиваются.

По способу задания области делятся на два типа:

a) гранично-определенные, задаваемые своей (замкнутой) границей такой, что коды пикселей границы отличны от кодов внутренней, перекрашиваемой части области. На коды пиксели внутренней части области налагаются два условия – они должны быть отличны от кода пикселей границы и кода пикселя перекраски. Если внутри гранично-определенной области имеется еще одна граница, нарисованная пикселями с тем же кодом, что и внешняя граница, то соответствующая часть области не должна перекрашиваться;
b) внутренне-определенные, нарисованные одним определенным кодом пикселя. При заполнении этот код заменяется на новый код закраски.

Заполняемая область или ее граница – некоторое связное множество пикселей. По способам доступа к соседним пикселям области делятся на 4- и 8-связные. В 4-связных областях доступ к соседним пикселям осуществляется в четырех направлениях - горизонтально влево и вправо и вертикально вверх и вниз. В 8-связных областях к этим направлениям добавляются еще 4 диагональных. Используя тот или иной тип связности, мы может, двигаясь от точки затравки, закрасить все пиксели области.

Важно отметить, что для 4-связной прямоугольной области граница 8-связна и наоборот у 8-ми связной области граница 4-х связна. Поэтому заполнение 4-связной области 8-связным алгоритмом может привести к «просачиванию» через границу и закраске пикселей в примыкающей области.

В общем, 4-связную область мы можем заполнить как 4-, так и 8-связным алгоритмом. Обратное же неверно.

Закраска области, заданной цветом границы

Рассмотрим алгоритмы закрашивания произвольного контура, который уже нарисован в растре. Сначала определяются координаты произвольного пикселя, находящегося внутри очерченного контура фигуры. Цвет этого пикселя изменяем на нужный цвет заполнения. Потом проводится анализ цветов всех соседних пикселей. Если цвет некоторого соседнего пикселя не равен цвету границы контура или цвету заполнения, то цвет этого пикселя изменяется на цвет заполнения. Потом анализируется цвет пикселей, соседних с предшествующими. И так далее, пока внутри контура все пиксели не перекрасятся в цвет заполнения.

Пиксели контура образуют границу, за которую нельзя выходить в ходе последовательного перебора всех соседних пикселей. Соседними могут считаться только четыре пикселя (сосед справа, слева, сверху и снизу четырехсвязность), или восемь пикселей (восьми-связность).

Простейший алгоритм закрашивания. Для всех алгоритмов закрашивания нужно задавать начальную точку внутри контура с координатами x_0, y_0. Простейший алгоритм можно описать всего двумя шагами.

Простейший рекурсивный алгоритм:

```
   void PixelFill (int x, int y, int border_color,
 int color)
        {
        int c = getpixel(x,y);
        if ((c!=border_color)&&(c!=color))
          {
          putpixel(x, y , color);
          PixelFill (x-1, y , border_color, color);
          PixelFill (x+1, y , border_color, color);
          PixelFill (x, y-1 , border_color, color);
          PixelFill (x, y+1 , border_color, color);
          }
         }
```

Этот алгоритм является слишком неэффективным, так как для всякого уже отрисованного пикселя функция вызывается ещё 4 раза и, кроме того, данный алгоритм не пригоден для закрашивания контуров фигур площадью в тысячу и

более пикселей, так как вложенные вызовы функций делаются для каждого пикселя, что приводит к переполнению стека в ходе выполнения программы.

Поэтому для решения задачи закраски области предпочтительнее алгоритмы, способные обрабатывать сразу целые группы пикселей, т. е. использовать их «связность». Если данный пиксель принадлежит области, то, скорее всего, его ближайшие соседи также принадлежат данной области. Группой таких пикселей обычно выступает полоса, определяемая правым пикселем.

Растровое заполнение круга и полигона

Его основная идея закрашивание фигуры отрезками прямых линий. Удобней использовать горизонтали. Алгоритм представляет собою цикл вдоль оси y, в ходе этого цикла выполняется поиск точек пересечения линии контура с соответствующими горизонталями.

Заполнение круга. Для заполнения круга можно использовать алгоритм вывода контура. В процессе выполнения этого алгоритма последовательно вычисляются координаты пикселей контура в границах одного октанта. Для заполнения следует выводить горизонтали, которые соединяют пары точек на контуре, расположенные симметрично относительно оси y (рис.1).

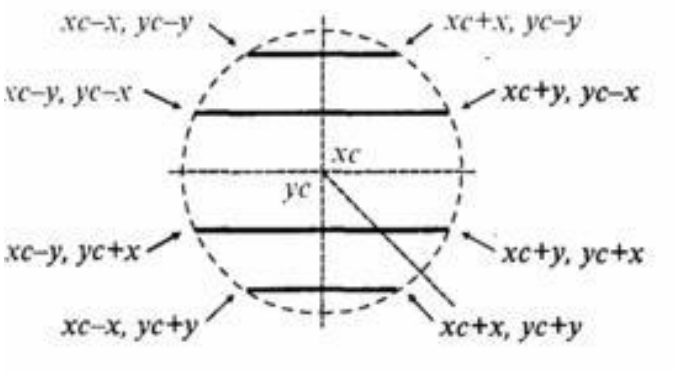


Рисунок 1 - Заполнение круга

```
void PixelFill_circle (int x0, int y0, int x, int y,
int color)
       {
        int col=getcolor();
        setcolor(color);
        line(x0+x,y0-y,x0-x,y0-y);
        line(x0-y,y0-x,x0+y,y0-x);
        line(x0-y,y0+x,x0+y,y0+x);
        line(x0-x,y0+y,x0+x,y0+y);
        setcolor(col);
        }
```

Так же можно создать и алгоритм заполнения эллипса.

Заполнение полигонов. Контур полигона определяется вершинами, которые соединены отрезками прямых ребрами.

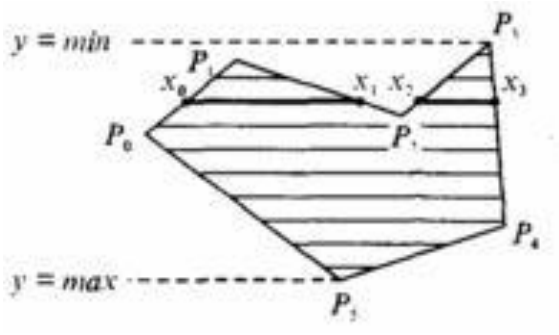


Рисунок 2 - Заполнение полигона

При нахождении точек пересечения горизонтали с контуром необходимо принимать во внимание особые точки. Если горизонталь имеет координату (y), совпадающую с координатой y_i вершины P_i тогда надлежит анализировать то, как горизонталь проходит через вершину. Если горизонталь при этом пересекает контур, как, например, в вершинах P_0 или P_4, то в массив записывается одна точка пересечения. Если горизонталь касается вершины контура (в этом случае вершина соответствует локальному минимуму или максимуму, как, например, в вершинах P_1, P_2, P_3 или P_5), тогда координата точки касания или не записывается, или записывается в массив два раза. Это является условием четного количества точек пересечения, хранящихся в массиве $\{x_j\}$.

Процедура определения точек пересечения контура с горизонталью, учитывая анализ на локальный максимум, может быть достаточно сложной.

Рассмотрим, какие случаи могут возникнуть при делении многоугольника на области сканирующей строкой.

1. Простой случай. Например, на рисунке 3 сканирующая строка $y = 4$ пересекает многоугольник при $x = 1$ и $x = 6$. Получается три области: $x < 1$; $1 \le x \le 6$; $x > 8$. Сканирующая строка $y = 6$ пересекает многоугольник при $x = 1$; $x = 2$; $x = 5$; $x = 6$. Получается пять областей: $x < 1$; $1 \le x \le 2$; $2 < x < 5$; $5 \le x \le 6$; $x > 6$. В этом случае x сортируется в порядке возрастания. Далее список иксов рассматривается попарно. Между парами точек пересечения закрашиваются все пикселы. Для $y = 4$ закрашиваются пикселы в интервале (1, 6), для $y = 6$ закрашиваются пикселы в интервалах (1, 2) и (5, 6).

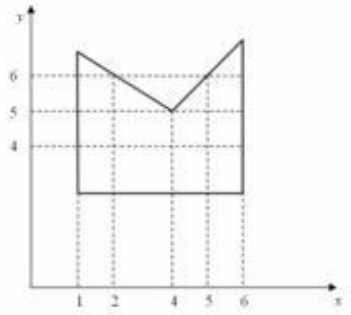


Рисунок 3 - Прохождение сканирующих строк по многоугольнику

2. Сканирующая строка проходит через вершину (рис. 4).

Например, по сканирующей строке y = 3 упорядоченный список x получится как (2, 2, 4). Вершина многоугольника была учтена дважды, и поэтому закрашиваемый интервал получается неверным: (2, 2). Следовательно, при пересечении вершины сканирующей строкой она должна учитываться единожды. И список по x в приведенном примере будет (2, 4).

3. Сканирующая строка проходит через локальный минимум или максимум (рис. 3 при y = 5). В этом случае учитываются все пересечения вершин сканирующей строкой. На рис. 5 при y = 5 формируется список x (1, 4, 4, 6). Закрашиваемые интервалы (1, 4) и (4, 6). Условие нахождения локального минимума или максимума определяется при рассмотрении концевых вершин для ребер, соединенных в вершине. Если у обоих концов координаты y больше, чем у вершины пересечения, то вершина – локальный минимум. Если меньше, то вершина пересечения – локальный максимум.

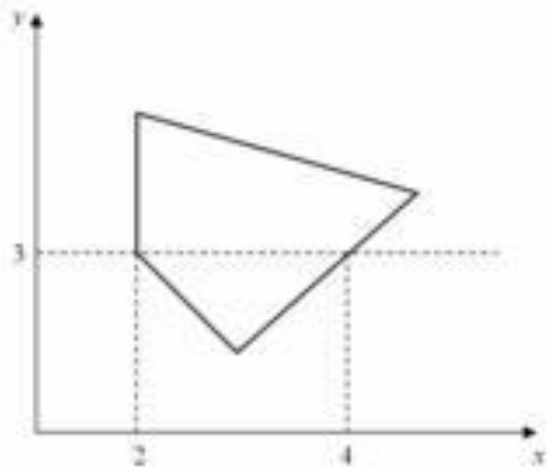


Рисунок 4 - Прохождение сканирующий строки через вершину

Для ускорения работы алгоритма используется список активных ребер (САР). Этот список содержит те ребра многоугольника, которые пересекают сканирующую строку. При пересечении очередной сканирующей строки вершины многоугольника, из САР удаляются ребра, которые находятся выше, и добавляются концы, которые пересекает сканирующая строка. При

работе алгоритма находятся пересечения сканирующей строки только с ребрами из САР.

Задание

Написать и отладить программу на языке C/C++ для вывода и заполнения объекта вывода (приложение). Для этого:

1. Написать необходимые функции для реализации растровых алгоритмов:
   - закраска области, заданной цветом;
   - закраска окружности (растровое заполнение);
   - закраска эллипса (растровое заполнение);
   - закраска многоугольника (растровое заполнение);
2. Выполнить задания по варианту согласно данным в приложении:
   - нарисовать и закрасить многоугольник;
   - нарисовать две окружности одна из них с полостью внутри;
   - закрасить окружности одну используя растровое заполнение, другую – затравочный метод;
   - нарисовать и закрасить эллипс;

Содержание отчета

Титульный лист.
Задание.
Описание реализованных методов.
Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

Лабораторная работа 7 Построение поверхностей

Цель работы: освоить алгоритмы построение поверхностей

Теоретическая часть

Остановим внимание на двух широко распространенных трехмерных представлениях поверхностей в пространстве. При описании и конструировании внешних геометрических форм в системах автоматизированного проектирования, и других приложениях машинной геометрии и графики различают два подхода. Первый – основан на методах точного аналитического описания кривых и поверхностей, второй – использует приближенные методы: метод интерполяции и метод аппроксимации.

Аналитической моделью будем называть описание поверхности математическими формулами. Можно использовать много разновидностей такого описания. Поверхности, задаваемые аналитически, представляют в виде неявной функции F(x, y, z) = 0 или в явном виде, например, в виде функции двух аргументов z = f(x, y) или параметрически.

Наиболее часто используется параметрическая форма описания поверхности. Запишем формулы для трехмерной декартовой системы координат (x, y, z):

$$x = F_x(s, t), \quad (1)$$
$$y = F_y(s, t), \quad (2)$$
$$z = F_z(s, t), \quad (3)$$

где s и t — параметры, которые изменяются в определенном диапазоне, а функции F_x, F_y, и F_z определяют форму поверхности.

Преимущества параметрического описания — легко описывать поверхности, которые соответствуют неоднозначным функциям, замкнутые поверхности. Описание можно сделать таким образом, что формула не будет существенно изменяться при поворотах поверхности, масштабировании.

Иногда удобно поверхность отобразить как многогранник, используя формулы аналитического описания для расчета координат вершин граней в процессе отображения.

Полигональная сетка представляет собой совокупность ребер, вершин и многоугольников. Вершины соединяются ребрами, а многоугольники рассматриваются как последовательности ребер или вершин. Сетку можно представить несколькими различными способами, каждый из них имеет свои достоинства и недостатки.

Для описания пространственных объектов здесь используются такие элементы: вершины, отрезки прямых (векторы), полилинии, полигоны, полигональные поверхности (рис. 1).

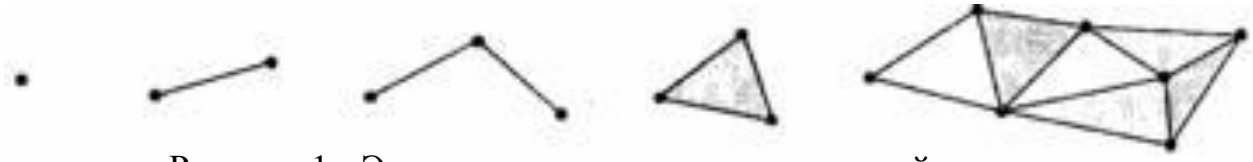


Рисунок 1 - Элементы векторно-полигональной модели

Элемент «вершина» (vertex) — главный элемент описания, все другие являются производными. При использовании трехмерной декартовой системы координат вершины определяются как (x,y,z). Каждый объект однозначно определяется координатами собственных вершин.

Вершина может моделировать отдельный точечный объект, размер которого не имеет значения, а также может использоваться в качестве конечной точки для линейных объектов и полигонов. Двумя вершинами задается вектор.

Несколько векторов составляют полилинию. Полилиния может моделировать отдельный линейный объект, толщина которого не учитывается, а также может представлять собой контур полигона.

Полигон моделирует площадный объект. Один полигон может описывать плоскую грань объемного объекта. Несколько граней составляют объемный объект в виде полигональной поверхности — многогранник или незамкнутую поверхность (в литературе часто употребляется название «полигональная сетка»).

Векторную полигональную модель можно считать наиболее распространенной в современных системах трехмерной компьютерной графики. Ее используют в системах автоматизированного проектирования, в компьютерных играх, в геоинформационных системах и т. п.

<u>Пример.</u> Построение сферы в виде многогранника, который аппроксимирует форму поверхности.

Уравнение сферы описывается формулами:

$x = a + R\cos u\cos v,$	(4)
$y = b + R\cos u\sin v,$	(5)
$z = c + R\sin u,$	(6)

a,b,c, – координаты центра.

```
void Draw(HWND hWnd)
{RECT rc;
HDC hdc;
point a,b,c,d;
POINT A[5];
int R=2, dl=10, db=10;
```

```
    hdc=GetDC(hWnd);
    if (hdc)    {
       GetClientRect(hWnd,&rc);
       Rectangle(hdc,0,0,rc.right,rc.bottom);
       for (int B=-90;B<=90; B+=db)
       {for (int L=0;L<=360; L+=dl)
               {
        a.x=R*cos(B*PI/180)*sin(L*PI/180); // 1 вершина
        a.z=R*cos(B*PI/180)*cos(L*PI/180);
        a.y=R*sin(B*PI/180);
        b.x=R*cos((B+db)*PI/180)*sin(L*PI/180); //2
        b.z=R*cos((B+db)*PI/180)*cos(L*PI/180);
        b.y=R*sin((B+db)*PI/180);
        c.x=R*cos((B+db)*PI/180)*sin((L+dl)*PI/180); //3
        c.z=R*cos((B+db)*PI/180)*cos((L+dl)*PI/180);
        c.y=R*sin((B+db)*PI/180);
        d.x=R*cos(B*PI/180)*sin((L+dl)*PI/180);  //4
        d.z=R*cos(B*PI/180)*cos((L+dl)*PI/180);
        d.y=R*sin(B*PI/180);
         A[0]=  MirToEk  (rc.right,rc.bottom,  a);    //
перевод в экранные координаты
         A[1]= MirToEk (rc.right,rc.bottom, b);
         A[2]= MirToEk (rc.right,rc.bottom, c);
         A[3]= MirToEk (rc.right,rc.bottom, d);
         A[4]= MirToEk (rc.right,rc.bottom, a);
         Polyline(hdc,A,5);
                }
       }
                          }
    ReleaseDC(hWnd,hdc);
    }
```

Задание

Написать и отладить программу на языке C/C++ для рисования поверхности по варианту (табл. И.1, приложение И). Для этого:

1. Реализовать функции для построения поверхности, заданной по варианту (приложение) в виде многогранника, который аппроксимирует форму поверхности.
2. Получить графическое изображение поверхности.

Содержание отчета

Титульный лист.
Задание.
Разработка и описание метода получения поверхности.
Текст программы.

При защите лабораторной работы тестирование программы **обязательно!**

ПРИЛОЖЕНИЕ

Варианты заданий

№	В виде неявной функции	Параметрическое представление	Описание
1	$\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1.$	$x = a\cos\theta\sin\phi,$ $y = b\sin\theta\sin\phi,$ $z = c\cos\phi.$ $0 \le \theta \le 2\pi,$ $0 \le \phi \le 2\pi,$	Эллипсоид. Величины a, b, c – полуоси эллипсоида. Если все они различны, эллипсоид называется трехосным В случае $a = b = c$ эллипсоид является сферой.
2	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=1;$	$x = a\cos\theta\,\mathrm{ch}\,\phi,$ $y = b\sin\theta\,\mathrm{sh}\,\phi,$ $z = c\,\mathrm{sh}\,\phi.$ $0 \le \theta \le 2\pi,$ $-\pi \le \phi \le \pi,$	Однополостной гиперболоид. в зависимости от соотношения параметров – трехосный ($a > b > c$), сжатый ($a = b > c$), вытянутый ($a = b < c$), равносторонний ($a = b = c$).
3	$\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=-1.$	$x = \pm a\,\mathrm{ch}\,\phi,$ $y = b\sin\theta\,\mathrm{sh}\,\phi,$ $z = c\cos\theta\,\mathrm{sh}\,\phi.$ $0 \le \theta \le 2\pi,$ $-\pi \le \phi \le \pi,$	Двухполостный гиперболоид. где a и b – мнимые полуоси; c – действительная полуось; в зависимости от соотношения параметров – трехосный ($a > b > c$), вращения ($a = b > c$, $a = b < c$), равносторонний ($a = b = c$).
4	$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$	$\begin{cases} x = u\sin\theta\cos v, \\ y = u\sin\theta\sin v, \\ z = H - u\cos\theta, \end{cases}$	Конус. В зависимости от значений полуосей a и b представляет собой эллиптический конус или круговой (при $a = b$). H – высота конуса; θ – угол между образующей и осью конуса; u, v – криволинейные координаты ($0 \le u \le H / \cos\theta$, $0 \le v \le 360$).
5	$x^2+y^2=R$	$x = r\cos v,\quad y = r\sin v,\quad z = u,$	Прямой круговой цилиндр. $0 \le u \le H$, $0 \le v \le 360$.
6	$2z=\frac{x^2}{p}+\frac{y^2}{q}$	$x = a\phi\cos\theta,$ $y = b\phi\sin\theta,$ $z = \phi^2.$ $0 \le \theta \le 2\pi,$ $0 \le \phi \le \phi_{max}.$	Эллиптический параболоид (при положительных p и q).
8	$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1.$	$x = a\cos\theta,$ $y = b\sin\theta,$ $z = \phi.$ $0 \le \theta \le 2\pi,$ $\phi_{min} \le \phi \le \phi_{max}$	Эллиптический цилиндр. Если a = b, то цилиндр оказывается круговым.

Лабораторная работа 8 Реализация алгоритмов растровой развертки линий

Цель работы: освоить алгоритмы растровой развертки линий

Теоретическая часть

Поскольку используемые дисплеи - растровые, для представления геометрического объекта на экране необходимо уметь получать его дискретное или растровое представление. Это означает, что заданной геометрической фигуре следует поставить в соответствие множество точек растра, которое в некотором смысле является приближением этой фигуры. Такое представление неоднозначно, т.к. существуют разные понятия и способы приближения непрерывного объекта. Например, растровым приближением круга на плоскости может служить множество внутренних точек круга, а прямой - множество точек - центров ячеек растра, с которыми эта прямая пересекается. Важным является эффективность того или иного представления.

Существуют стандартные процедуры программной или аппаратной реализации алгоритмов генерации линий и закраски многоугольников в различных графических пакетах, но для студентов направления 09.03.01 необходимо представлять и уметь программно реализовывать такие алгоритмы при работе с устройствами типа дисплея, принтера, плоттера. Необходимость использовать свою версию алгоритма генерации возникает, например, когда атрибуты инициируемого пиксела зависят от каких-либо условий (положения пиксела на прямой или внутри закрашиваемого многоугольника) или когда Вас не устраивает скорость работы "стандартного" алгоритма. Такие ситуации типичны при реализации алгоритмов загораживания (например, растровой реализации одной из версий алгоритма плавающего горизонта или метода буфера глубины).

Разложение отрезка в растр

Процесс последовательной инициализации множества пикселов экрана, составляющих отрезок, называется его растровой разверткой, а само это множество - растровым представлением отрезка. Далее будем рассматривать два таких представления: восьмисвязное и четырехсвязное. Четырехсвязная развертка отрезка предполагает различие у соседних точек на отрезке только значения одной координаты. У восьмисвязной - могут отличаться обе координаты.

Простой пошаговый алгоритм. Если увеличивать с определенным шагом координату X, а затем находить координату Y используя уравнение прямой Y=mX+b и подкрашивать пиксел с координатами (X,ROUND(Y)), то потребуется много времени (на нецелочисленные операции). Если шаг по X принять равным единице, то m=dY/dX сводится к m=dY, т.е.изменение X на единицу приведет к изменению углового коэффициента на m. Т.о., если

X(I+1)=X(I)+1,то Y(I+1)=Y(I)+m. Алгоритм корректно работает только для отрезков в первом и восьмом октандах. В остальных случаях требует модификации, что предлагается сделать студентам самостоятельно в ходе выполнения данной лабораторной работы. При модификации следует учесть, что при m>1,единичный шаг по X приведет к такому увеличению Y, при котором две соседние точки на прямой расположатся далеко друг от друга. Поэтому X и Y следует поменять, чтобы увеличивать на единицу Y, а X - на dX = dY/m = 1/m.

Описание алгоритма для работы в первом и восьмом октандах:

Дано : m - угловой коэффициент(-1<m<1);

(x1,y1) (x2,y2) - координаты концов отрезка;

col - цвет для подкрашивания точек отрезка;

Нарисовать_точку (x,y,col) - процедура для подкрашивания цветом col пиксела с координатами(x,y);

Round(x) - функция округления x до ближайшего целого.

Требуется : разложить отрезок в растр.

1.Если x1 # x2 идти к 2 иначе к 4.

2.m=(y2-y1)/(x2-x1); y=y1;

3.Для x от x1 до x2 выполнить:

начало

Нарисовать_точку(x,Round(y),col);

y=y+m;

конец

4.Если y1=y2 идти к 5 иначе к 6.

5.Нарисовать_точку(x1,y1,col). Идти к 7.

6.Вывести сообщение об ошибке (вертикаль).

7.Закончить.

Целочисленные алгоритмы Брезенхема

Основной недостаток простейшего алгоритма – использование вещественных вычислений. В 1965 году Брезенхемом был предложен целочисленный алгоритм растрового построения отрезка, первоначально предназначенный для графопостроителей. Суть алгоритма в следующем: в процессе работы одна из координат либо x, либо y (в зависимости от углового коэффициента) изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния - (e) между действительным положением отрезка и ближайшими координатами растра (e назовем управляющей переменной). Алгоритм построен так, что на каждом шаге проверяется лишь знак e и корректируется ее значение после каждого изменения очередной координаты. Значение исходной управляющей переменной:

e=2*(y2-y1)-(x2-x1),

где x1,y1,x2,y2 - координаты начальной и конечной точек отрезка. В каждом шаге при e>=0 значение y от предыдущего увеличивается на единицу, а e уменьшается на 2*(x2-x1), в противном случае - y не меняется, а значение e увеличивается на 2*(y2-y1). В обоих случаях координата x следующего пиксела увеличивается на единицу от предыдущего значения.

Описание общего алгоритма Брезенхема:

Для восьмисвязной развертки:

Дано: (x1,y1,x2,y2)-координаты начальной и конечной точек отрезка; резка

col - цвет пикселов отрезка;

Нарисовать_точку(x,y,col) - процедура для подкрашивания цветом col пиксела с координатами(x,y);

sign(x) - функция, возвращающая 1,если x>=0 и -1,если x<0;

round(x)- функция округления x до ближайшего целого;

abs(x) - функция, возвращающая целое от x.

Требуется : разложить отрезок в растр.

1.Присвоить: x=x1; y=y1; dx=abs(x2-x1); dy=abs(y2-y1);

s1=sign(x2-x1); s2=sign(y2-y1);

2.Если dy>dx идти к 3, иначе к 4.

3.Поменять ролями dx и dy.Переменной l присвоить значение "истина". Идти к 5.

4.l присвоить значение "ложь". Идти к 5.

5.Присвоить e исходное значение 2*dy-dx.

6.Для i от 1 до dx выполнить:

начало

Нарисовать_точку(x,y,col); Пока e>=0 выполнить:

начало если l="истина" то x=x+s1 иначе y=y+s2;

e=e-2*dx

конец

Если l="истина" то y=y+s2 иначе x=x+s1; e=e+2*dy

конец

7.Нарисовать_точку(x,y,col).

8.Закончить.

Для четырехсвязной развертки:

Дано: (x1,y1,x2,y2)-координаты начальной и конечной точек отрезка;

col - цвет пикселов отрезка;

Нарисовать_точку (x,y,col) - процедура для подкрашивания цветом col пиксела с координатами(x,y);

sign(x) - функция, возвращающая 1,если x>=0 и -1,если x<0;

round(x) - функция округления x до ближайшего целого;

abs(x) - функция, возвращающая целое от x;

Требуется : разложить отрезок в растр.

1.Присвоить: x=x1; y=y1; dx=abs(x2-x1); dy=abs(y2-y1);

s1=sign(x2-x1); s2=sign(y2-y1);

2.Если dy<dx то присвоить переменной l значение "ложь" и идти к 4 иначе идти к 3.

3.Поменять ролями dx и dy. Переменной l присвоить значение "истина". Идти к 4.

4.Присвоить e исходное значение 2*dy-dx.

5.Для i от 1 до dx+dy выполнить:

начало

Нарисовать_точку(x,y,col);

Если e<0 выполнить: начало

если l="истина" то y=y+s2 иначе x=x+s1;

e=e+2*dy

конец иначе выполнить:

начало

если l="истина" то x=x+s1 иначе y=y+s2;

e=e-2*dx

конец

конец

6.Нарисовать_точку(x,y,col).

7.Закончить.

Растровая развертка окружностей

Существуют несколько простых способов преобразования окружности в растровую форму. Например, по формуле X*X+Y*Y=R*R для окружности с центром в начале координат. Чтобы изобразить четверть такой окружности, на каждом шаге следует поменять X от 0 до R на единицу и вычислить Y как SQRT(R*R-X*X). Остальные четверти изображают симметрично. Этот метод содержит операции умножения и извлечения корня, потому не эффективен. Кроме того, при X близких к R, в окружности появляются заметные промежутки, так как при таких X тангенс угла наклона касательной к окружности стремится к бесконечности. Процесс можно улучшить, если вычислять одну восьмую часть окружности, а остальные семь частей отображать симметрично (в предыдущем случае X менять от 0 до R/SQRT(2)). Но необходимый эффект можно получить только при работе с целыми числами (алгоритмами, подобными рассмотренным выше). Для генерации окружностей Брезенхем предложил следующий алгоритм:

Дано: rad - радиус окружности

col - цвет пикселов окружности

e - управляющая переменная

Нарисовать_точку(x,y,col) - процедура для подкрашивания цветом col пиксела с координатами(x,y)

Требуется: разложить окружность в растр.

1.Присвоить:x=0;y=rad;e=3-2*rad;

2.Пока x<y выполнить:

Нарисовать_точку(x,y,col);

Нарисовать_точку(y,x,col);

Нарисовать_точку(y,-x,col);

Нарисовать_точку(x,-y,col);

Нарисовать_точку(-x,-y,col);

Нарисовать_точку(-y,-x,col);

Нарисовать_точку(-y,x,col);

Нарисовать_точку(-x,y,col)

3.Если e<0 то e=e+4*x+6 и идти к 5 иначе идти к 4.

4.Выполнить: e=e+4*(x-y)+10; y=y-1

5.x=x+1 идти к 2 пока x<y иначе к 6.

6.Если x=y то идти к 7 иначе к 8.

7.Выполнить:

Нарисовать_точку(x,y,col);

Нарисовать_точку(y,x,col);

Нарисовать_точку(y,-x,col);

Нарисовать_точку(x,-y,col);

Нарисовать_точку(-x,-y,col);

Нарисовать_точку(-y,-x,col);

Нарисовать_точку(-y,x,col);

Нарисовать_точку(-x,y,col).

8.Закончить.

Задание на лабораторную работу

1. Написать программу, реализующую алгоритмы построения прямой: простой пошаговый алгоритм и алгоритмы Брезенхема для четырех- и восьмисвязной развертки.

2. Проверить правильность работы программы, нарисовав, например, каждым алгоритмом семейство радиальных прямых, выходящих из одной точки с шагом 15 градусов.

3. Написать и отладить программу, реализующую два алгоритма построения окружности: по формуле Y=+-SQRT(r*r-x*x) и Брезенхема. В обоих случаях использовать свойство симметрии окружности (в первом - найдя точки четверти окружности, остальные - отразив симметрично; во втором - свойство симметрии использовать полностью).

Дополнительное задание

Используя программы работы с мышью и программы, написанные при выполнении пунктов 1-3 задания, вывести на экран простой рисунок из линий и окружностей.

Пример программы работы с мышью:

{

Пример использования мыши в программах на языке Паскаль. Работу с мышью обеспечивает драйвер, отслеживающий перемещения ее курсора, нажатие и отпускание кнопок. Реализация работы идет через механизм прерываний. Номер прерывания - 33h. В регистры помещаются входные и выходные параметры. В языке Паскаль для этих целей используют процедуру Intr(IntNo:Byte;varRegs:Registers) модуля Dos. Нельзя выводить изображение поверх курсора мыши. В случае необходимости нужно убрать курсор, вывести изображение, затем вывести курсор.

}

```
program mouse;
uses Crt,Graph,Dos;
var
grDriver: Integer;
grMode: Integer;
ErrCode: Integer;
X,Y,X1,Y1:Integer;
L,M,R:Boolean;
BL:Boolean;
```

{

Инициализация и проверка наличия мыши. Возвращает истину (есть) или ложь (нет) }

```
function ReadMouse : Boolean;
var
r: Registers;
begin
r.ax:=0;
intr($33,r);
ReadMouse:=r.ax=$ffff;
end;
{Визуализировать курсор мыши}
procedure ShowMouseCursor;
var
r:registers;
begin
r.ax:=1;
intr($33,r);
end;
{Сделать курсор мыши невидимым (перемещение отслеживается)}
procedure HideMouseCursor;
```

```
var
r:registers;
begin
r.ax:=2;
intr($33,r);
end;
{Возврат координат положения курсора и нажатой клавиши мыши}
procedure ReadMouseStatus(var x,y:integer;var LB,MB,RB :boolean);
var
r:Registers;
begin
r.ax:=3;
intr($33,r);
x:=r.cx;
y:=r.dx;
LB:=(r.bx And 1) <> 0;
RB:=(r.bx And 2) <> 0;
MB:=(r.bx And 4) <> 0;
end;
{Перемещение курсора мыши к указанным координатам}
procedure MoveMouseCursor(x,y:integer);
var r:registers;
begin
r.ax:=4;
r.cx:=x;
r.dx:=y;
intr($33,r);
end;
{
Основная программа. Рисует линию по двум точкам, задаваемым
щелчком левой кнопки мыши. По щелчку правой кнопки мыши - выходит
из программы
}
begin
grDriver := detect;
InitGraph(grDriver, grMode,'d:\tp7\bgi');
ErrCode := GraphResult;
if ErrCode = grOk then
begin { Do graphics }
L:=false;M:=false;R:=false;Bl:=true;
SetColor(Cyan);
```

```
SetBKColor(LightGray);
if ReadMouse then begin
ShowMouseCursor;
While true do
begin
Repeat ReadMouseStatus (X,Y,L,M,R) Until L or R;
If R then begin
CloseGraph;
Exit end;
While L do ReadMouseStatus(X,Y,L,M,R);
if BL then
begin
X1:=X;Y1:=Y;
Bl:=false
end
else
begin
HideMouseCursor;
Line(X1,Y1,X,Y);
ShowMouseCursor;
Bl:=true
end;
end;
end
else
begin
RestoreCrtMode;Writeln('error');Halt;
end;
end
else
Writeln('Graphics error:', GraphErrorMsg(ErrCode));
end.
```

Требования к защите лабораторной работы

Защита лабораторной работы состоит из демонстрации преподавателю результатов выполнения заданий на лабораторную работу и ответов на задаваемые преподавателем вопросы по ходу демонстрации.

Требования к отчету

Содержит титульный лист, оформленный согласно требованиям кафедры, и листинг программы - результат выполнения заданий лабораторной работы с обязательными комментариями.

Вопросы для самоподготовки

1. Каков визуальный эффект применения простого пошагового алгоритма к отрезкам, угловой коэффициент которых больше единицы или меньше минус единицы?

2. Как избавиться от этого эффекта?

3. Почему простой пошаговый алгоритм не используют для генерации отрезков в современных графических пакетах?

4. Каков визуальный эффект при нахождении координаты Y всех точек или одной четверти окружности по формуле X^2+Y^2=R^2? Как избавиться от этого эффекта?

5. Каким образом в алгоритмах Брезенхема удается избежать нежелательных эффектов при построении точек линий?

Лабораторная работа 9 Реализация преобразований по отношению к плоской фигуре

Цель работы: овоить основные преобразования на плоскости: параллельный перенос, поворот, растяжение/сжатие, зеркальное отражение. Получить практические навыки реализации описанных преобразований по отношению к плоской фигуре, заданной набором вершин и соединяющих их отрезков. Получить практические навыки разработки приложений в среде Microsoft Visual Studio.

Задание и методические указания к выполнению работы

Задание на лабораторную работу

Написать графическое приложение Win32 на языке C++, выполняющее действия, описанные в конкретном задании. Реализовать в программе корректную обработку ошибок при вызове любых API функций: в случае возникновения ошибки выдать код ошибки, имя вызвавшей ошибку функции и текстовое описание ошибки, после чего завершить работу программы.

Программа должна быть реализована в виде проекта в среде разработки Microsoft Visual Studio или Microsoft Visual Studio .Net. Программы, выполненные на языках программирования, отличных от C++ или в иной среде разработки не принимаются.

Исходный код программы должен быть читабельным и включать в себя комментарии.

При защите лабораторной работы студент должен отчитаться по выполненной работе.

Выполненные задания оцениваются по следующим критериям:

- корректность реализации основного задания;
- читабельность исходного кода программы и наличие в нем комментариев;
- умение автора программы объяснить алгоритм ее работы и исходный код.
- дополнительная функциональность, реализованная помимо необходимого задания.

По каждому критерию выставляется оценка по пятибальной шкале. Общая оценка за выполненное задание является суммой оценок по каждому критерию. Оценка за выполненное задание влияет на сдачу экзамена по дисциплине. Работа считается не сданной, если оценка за один из первых трёх критериев составляет ниже трех баллов.

Форма отчетности

1. Отчет по лабораторной работе, который включает в себя следующие разделы: постановка задачи, метод решения, текст программы, результаты работы программы в виде снимков экрана (не менее трех). Отчет сдается преподавателю в печатном и электронном виде.
2. Рабочая копия исходного кода программы, включая файлы проекта среды разработки. Сдается в электронном виде.

3. Устная беседа по выполненной работе, в ходе которой задаются практические вопросы по исходному коду программы и теоретические вопросы по теме лабораторной работы.

Методические указания к выполнению работы

Для выполнения работы необходимо:

1. Открыть среду разработки Microsoft Visual Studio.
2. Создать новый проект консольного приложения Win32.
3. Набрать исходный код программы в текстовом редакторе среды разработки.
4. Откомпилировать программу. В случае возникновения ошибок компиляции исправить их.
5. Отладить программу с использованием встроенного отладчика среды разработки.
6. Проверить корректность работы программы при различных ситуациях.
7. Составить отчет о проделанной работе в текстовом редакторе Microsoft Word.

Контрольные вопросы

1. Опишите координатную систему на плоскости.
2. Напишите и поясните общее уравнение прямой. Какие частные уравнения прямой вы знаете?
3. Запишите в матричном виде преобразования параллельного переноса, вращения, зеркального отражения, растяжения (сжатия).

Варианты заданий на лабораторную работу

Общее задание – применение описанных в разделе цели работы преобразований на плоскости к плоской фигуре, заданной набором вершин и соединяющих их отрезков. Программа должна содержать главное окно, подобное представленному на рис. 1. В рабочей области окна должна отображаться плоская фигура с примененными к ней преобразованиями. Сами преобразования задаются пользователем при помощи графических элементов интерфейса, например, меню или панели инструментов. При каждом выборе пункта меню или нажатии кнопки на панели инструментов, к фигуре должно быть применено соответствующее преобразование, после чего фигура должна быть перерисована в рабочей области окна. Например, можно создать кнопки «Поворот на 10^0 по часовой стрелке», «Поворот на 10^0 против часовой стрелки», «Сжатие на 10%», «Растяжение на 10%» и т.д. Т.е. каждому преобразованию соответствует некоторый набор кнопок, каждая из которых определяет фиксированные параметры этого преобразования. В программе в

обязательном порядке должны использоваться классы матриц и векторов, а не их скалярные представления в виде переменных x, y и т.п.

Индивидуальные задания соответствуют реализации общего задания, но с различными фигурами для каждого студента. Задания нескольких студентов с одной и той же фигурой не принимаются.

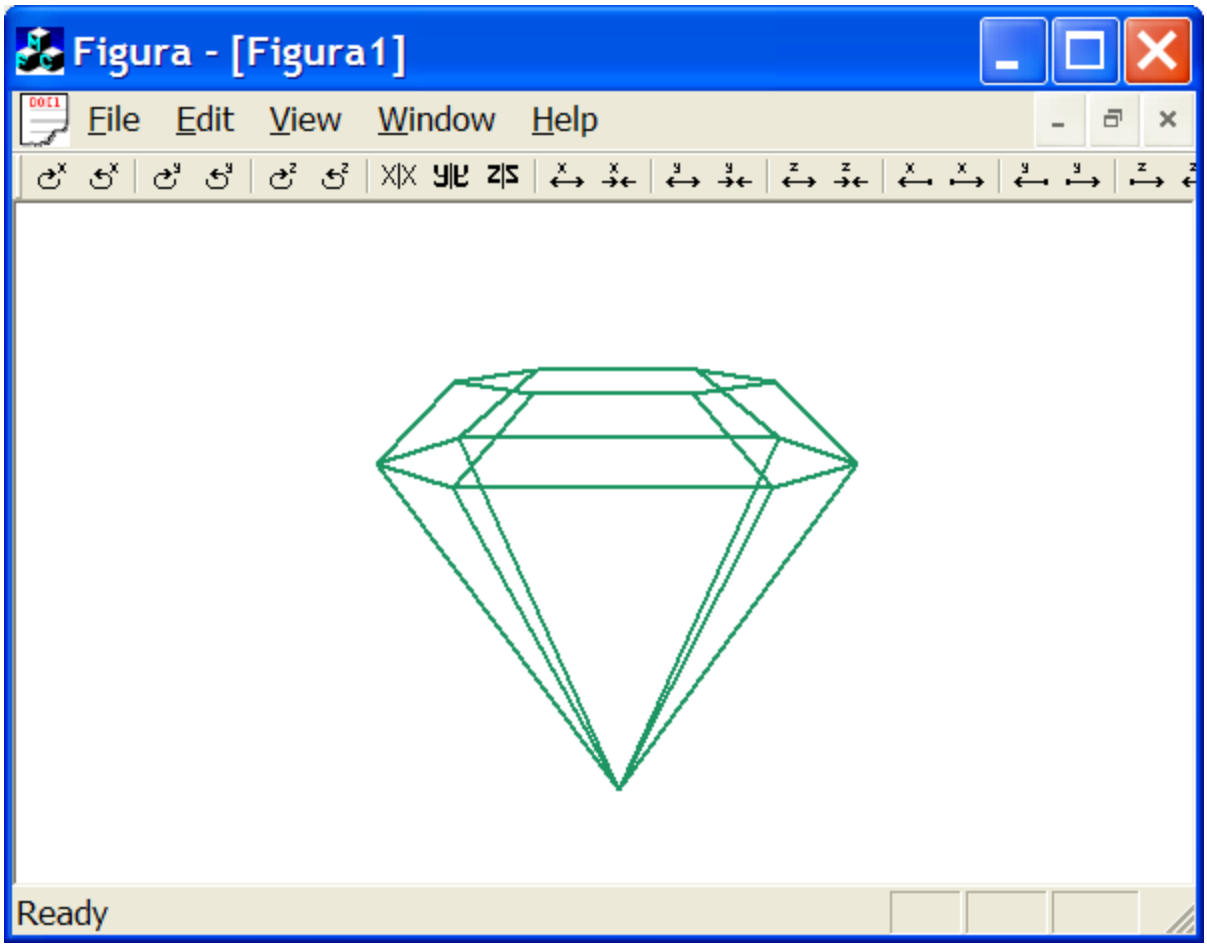


Рисунок 1 - Пример главного окна программы

Лабораторная работа 10 Реализация преобразований по отношению к объемной фигуре

Цель работы: изучить основные преобразования в пространстве: параллельный перенос, поворот, растяжение/сжатие, зеркальное отражение. Получить практические навыки реализации описанных преобразований по отношению к объемной фигуре, заданной набором вершин и граней. Получить практические навыки разработки приложений в среде Microsoft Visual Studio

Задание и методические указания к выполнению работы

Задание на лабораторную работу

Написать графическое приложение Win32 на языке C++, выполняющее действия, описанные в конкретном задании. Реализовать в программе корректную обработку ошибок при вызове любых API функций: в случае возникновения ошибки выдать код ошибки, имя вызвавшей ошибку функции и текстовое описание ошибки, после чего завершить работу программы.

Программа должна быть реализована в виде проекта в среде разработки Microsoft Visual Studio или Microsoft Visual Studio .Net. Программы, выполненные на языках программирования, отличных от C++ или в иной среде разработки не принимаются.

Исходный код программы должен быть читабельным и включать в себя комментарии.

При защите лабораторной работы студент должен отчитаться по выполненной работе.

Выполненные задания оцениваются по следующим критериям:

- корректность реализации основного задания;
- читабельность исходного кода программы и наличие в нем комментариев;
- умение автора программы объяснить алгоритм ее работы и исходный код.
- дополнительная функциональность, реализованная помимо необходимого задания.

По каждому критерию выставляется оценка по пятибальной шкале. Общая оценка за выполненное задание является суммой оценок по каждому критерию. Оценка за выполненное задание влияет на сдачу экзамена по дисциплине. Работа считается не сданной, если оценка за один из первых трёх критериев составляет ниже трех баллов.

Форма отчетности

1. Отчет по лабораторной работе, который включает в себя следующие разделы: постановка задачи, метод решения, текст программы, результаты работы программы в виде снимков экрана (не менее трех). Отчет сдается преподавателю в печатном и электронном виде.
2. Рабочая копия исходного кода программы, включая файлы проекта среды разработки. Сдается в электронном виде.

3. Устная беседа по выполненной работе, в ходе которой задаются практические вопросы по исходному коду программы и теоретические вопросы по теме лабораторной работы.

Методические указания к выполнению работы

Для выполнения работы необходимо:

1. Открыть среду разработки Microsoft Visual Studio.
2. Создать новый проект консольного приложения Win32.
3. Набрать исходный код программы в текстовом редакторе среды разработки.
4. Откомпилировать программу. В случае возникновения ошибок компиляции исправить их.
5. Отладить программу с использованием встроенного отладчика среды разработки.
6. Проверить корректность работы программы при различных ситуациях.
7. Составить отчет о проделанной работе в текстовом редакторе Microsoft Word.

Контрольные вопросы

1. Опишите координатную систему в пространстве.
2. Напишите и поясните общее уравнение плоскости. Как можно вычислить нормаль к плоскости.
3. Запишите в матричном виде преобразования параллельного переноса, вращения, зеркального отражения, растяжения (сжатия).

Варианты заданий на лабораторную работу

Общее задание соответствует лабораторной работе 1 с той разницей, что вместо плоской фигуры используется объемная. Соответственно, расширяется набор вариантов одного и того же преобразования: параллельный перенос вдоль одной из трех координатных осей; поворот вокруг одной из трех координатных осей по часовой и против часовой стрелки; растяжение/сжатие вдоль одной из трех координатных осей; зеркальное отражение относительно одной из трех координатных плоскостей.

Лабораторная работа 11 Возможности интерфейса программирования, предоставляемого стандартом OpenGL

Цель работы: изучение основ программирования графики с использованием библиотеки трехмерной графики и моделирования OpenGL.

Теоретическая часть

Возможности OpenGL

OpenGL – Open Graphics Library, открытая графическая стандартная библиотека для программирования трехмерной графики для многих 32-разрядных операционных систем (Windows, Linux в том числе).

В отличие от Direct3D, которая характерна только для Windows, OpenGl содержит в себе более 250 процедур и функций для построения 3D графики и рендеринга. Они находятся в opengl32.dll (Windows\ system\) и в расширении glu32.dll.

К основным возможностям OpenGL можно отнести:

- *геометрические* (точки, линии, полигоны) *и растровые* (битовый массив (bitmap) и образ (image)) *примитивы*;
- *использование B-сплайнов* для рисования кривых по опорным точкам;
- *альфа-канал***.** Позволяет делать предметы прозрачными, уровень прозрачности от 0 до 100 %;
- *антиалиасинг (сглаживание)* цветовых переходов для получения более реалистического изображения;
- *буфер аккумулятора***.** Дополнительный буфер для спецэффектов и глобального сглаживания по всей сцене;
- *градиентная заливка* полигонов и отрезков;
- *двойная буферизация***.** Для устранения мерцания при мультипликации. Изоб-ражение каждого кадра сначала рисуется во втором (невидимом) буфере, а потом, когда кадр полностью нарисован, весь буфер отображается на экране;
- *заливка и освещенность фактур***.** К фактурам применяются эффекты освещенности и затенения в зависимости от характеристик «материала» ;
- *пространственные преобразования***.** Масштабирование, вращение и перемещение объектов в пространстве;
- *текстуры (меппинг)* Наложение двухмерных изображений на объемные поверхности для придания сцене реализма;
- *атмосферные эффекты,* такие как туман, дым, дымка делают изображения, созданные компьютером, более реалистичными.

OpenGL позволяет:

1) создавать объекты из геометрических примитивов (точки, линии, грани и битовые изображения).

2) располагать объекты в трёхмерном пространстве и выбирать способ и параметры проецирования.

3) вычислять цвет всех объектов. Цвет может быть как явно задан, так и вычисляться с учётом источников света, параметров освещения, текстур.

4) переводить математическое описание объектов и связанной с ними информации о цвете в изображение на экране.

При этом OpenGL может осуществлять дополнительные операции, такие, как удаление невидимых фрагментов изображения.

Основные типы данных OpenGL

Все команды (процедуры и функции) OpenGL начинаются с префикса **gl**, а все константы – с префикса **GL_**. Кроме того, в имена функций и процедур OpenGL входят суффиксы, несущие информацию о числе передаваемых параметров и о их типе. В табл. 1 приводятся вводимые OpenGL типы данных, стандартные типы языка C, которым они соответствуют, и суффиксы, которым они соответствуют.

Некоторые команды OpenGL оканчиваются на букву v. Это говорит о том, что команда получает указатель на массив значений, а не сами эти значения в виде отдельных параметров. Многие команды имеют как векторные, так и не векторные версии. Например, конструкции:

```
glColor3f(1.0, 1.0, 1.0);
```

и

```
GLfloat color[] = {1.0, 1.0, 1.0};
glColor3fv(color);
```

эквивалентны.

Таблица 1 - Типы данных OpenGL

Суффикс	Описание	Тип в C	Тип в OpenGL
b	8-битовое целое	char	GLbyte
s	16-битовое целое	short	GLshort
i	32-битовое целое	long	GLint GLsizei
f	32-битовое вещественное число	float	GLfloat, GLclampf
d	64-битовое вещественное число	double	GLdouble, GLclampd
ub	8-битовое беззнаковое целое	unsigned char	GLubyte, GLboolean
us	16-битовое беззнаковое целое	unsigned short	GLushort
ui	32-битовое беззнаковое целое	unsigned long	GLuint, GLenum, GLbitfield

OpenGL можно рассматривать как автомат, находящийся в одном из нескольких состояний. Внутри OpenGL содержится целый ряд переменных, например, текущий цвет или текущий режим закрашивания. Если установить

текущий цвет, то все последующие объекты будут этого цвета до тех пор, пока текущий цвет не будет изменён.

По умолчанию каждая системная переменная имеет своё значение, и в любой момент значение каждой из этих переменных можно узнать. Обычно для этого используется одна из следующих функций: glGetBooleanv(), glGetDoublev(), glGetFloatv() и glGetIntegerv(). Для определения значений некоторых переменных служат специальные функции.

Рисование геометрических объектов

Работа с буферами и задание цвета объектов

OpenGL содержит внутри себя несколько различных буферов. Среди них фрейм буфер (где строится изображение), z-буфер, служащий для удаления невидимых поверхностей, буфер трафарета и аккумулирующий буфер.

Для очистки внутренних буферов служит процедура glClear(GLbitfield mask), очищающая буферы, заданные переменной mask. Параметр mask является комбинацией следующих констант:

GL_COLOR_BUFFER_BIT – очистить буфер изображения (фреймбуфер);

GL_DEPTH_BUFFER_BIT – очистить z-буфер;

GL_ACCUM_BUFFER_BIT – очистить аккумулирующий буфер;

GL_STENCIL_BUFFER_BIT – очистить буфер трафарета.

Цвет, которым очищается буфер изображения, задаётся процедурой glClearColor(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha). Значение, записываемое в z-буфер, при очистке задаётся процедурой glClearDepth(GLfloat depth). Значение, записываемое в буфер трафарета, при очистке задаётся процедурой glClearStencil(GLint s). Цвет, записываемый в аккумулирующий буфер, при очистке задаётся процедурой glClearAccum(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha).

Сама команда glClear очищает одновременно все заданные буферы, заполняя их соответствующими значениями.

Для задания цвета объекта служит процедура:

glColor{3 4}{b s i f d ub us ui}[v](TYPE red, …).

Цифра 3 или 4 указывает на количество требуемых аргументов, а буква, следующая за цифрой, показывает тип аргументов. Например, в процедуру glColor3i будут переданы три параметра целого типа.

Если значение параметра не задано, то оно автоматически полагается равным единице. Версии процедуры glColor, где параметры являются переменными с плавающей точкой, автоматически обрезают переданные значения в отрезок [0, 1].

Процедура glFlush() вызывает немедленное рисование ранее переданных команд. При этом ожидания завершения всех ранее переданных команд не

происходит. С другой стороны, команда glFinish() ожидает, пока не будут завершены все ранее переданные команды.

Если нужно включить удаление невидимых поверхностей методом z-буфера, то z-буфер необходимо очистить и передать команду glEnable(GL_DEPTH_TEST). Команду glEnable() можно выполнить только один раз при инициализации системных переменных OpenGL. Очистку z-буфера необходимо производить перед началом построения очередного кадра изображения.

Задание графических примитивов

Все геометрические примитивы в OpenGL задаются вершинами (набором чисел, определяющих их координаты в пространстве).

OpenGL работает с однородными координатами (x, y, z, w). Если координата z не задана, то она считается равной нулю. Если координата w не задана, то она считается равной единице.

Под линией в OpenGL подразумевается отрезок, заданный своими начальной и конечной вершинами.

Под гранью (многоугольником) в OpenGL подразумевается замкнутый выпуклый многоугольник с несамопересекающейся границей.

Все геометрические объекты в OpenGL задаются посредством вершин, а сами вершины задаются процедурой:

glVertex{2 3 4}{s i f d}[v](TYPE x, …),

где реальное количество параметров определяется первым суффиксом (2, 3 или 4), а суффикс **v** означает, что в качестве единственного аргумента выступает массив, содержащий необходимое количество координат. Например:

```
glVertex2s(1, 2);
glVertex3f(2.3, 1.5, 0.2);
GLdouble vect[] = {1.0, 2.0, 3.0, 4.0};
glVertex4dv(vect);
```

Для задания геометрических примитивов необходимо как-то выделить набор вершин, определяющих этот объект. Для этого служат процедуры glBegin() и glEnd(). Процедура glBegin(GLenum mode) обозначает начало списка вершин, описывающих геометрический примитив. Тип примитива задаётся параметром mode, который принимает одно из следующих значений:

GL_POINTS – набор отдельных точек;

GL_LINES – пары вершин, задающих отдельные точки;

GL_LINE_STRIP – незамкнутая ломаная;

GL_LINE_LOOP – замкнутая ломаная;

GL_POLYGON – простой выпуклый многоугольник;

GL_TRIANGLES – тройки вершин, интерпретируемые как вершины отдельных треугольников;

GL_TRIANGLE_STRIP – связанная полоса треугольников;

GL_TRIANGLE_FAN – веер треугольников;

GL_QUADS – четвёрки вершин, задающие выпуклые четырёхугольники;

GL_QUAD_STRIP – полоса четырёхугольников.

Процедура glEnd() отмечает конец списка вершин.

Между командами glBegin() и glEnd() могут находится команды задания различных атрибутов вершин glVertex(), glColor(), glNormal(), glCallList(), glCallLists(), glTexCoord(), glEdgeFlag(), glMaterial(). Между командами glBegin() и glEnd() все остальные команды OpenGL недопустимы и приводят к возникновению ошибок.

Рассмотрим в качестве примера задание окружности:

```
glBegin(GL_LINE_LOOP);
for (int i = 0; I < N; i++)
{
float angle = 2 * M_PI * i / N;
glVertex2f(cos(angle), sin(angle));
}
glEnd();
```

Хотя многие команды могут находится между glBegin() и glEnd(), вершины генерируются при вызове glVertex(). В момент вызова glVertex() OpenGL присваивает создаваемой вершине текущий цвет, координаты текстуры, вектор нормали и т. д. Изначально вектор нормали полагается равным (0, 0, 1), цвет полагается равным (1, 1, 1, 1), координаты текстуры полагаются равными нулю.

Рисование точек, линий и многоугольников

Для задания размеров точки служит процедура glPointSize(GLfloat size), которая устанавливает размер точки в пикселях, по умолчанию он равен единице.

Для задания ширины линии в пикселях служит процедура gflineWidth(GLfloatwidth). Шаблон, которым будет рисоваться линия, можно задать при помощи процедуры glLineStipple(Glint factor, GLushort pattern).

Шаблон задается переменной pattern и растягивается в factor раз. Использование шаблонов линии необходимо разрешить при помощи команды glEnable(GL_UNE_STIPPLE). Запретить использование шаблонов линий можно командой glDisable(GL_LINE_ST1PPLE).

Многоугольники рисуются как заполненные области пикселей внутри границы, хотя их можно рисовать либо только как граничную линию, либо просто как набор граничных вершин.

Многоугольник имеет две стороны, лицевую и нелицевую, и может быть отрисован по-разному в зависимости от того, какая сторона обращена к наблюдателю. По умолчанию обе стороны рисуются одинаково. Для задания

того, как именно следует рисовать переднюю и заднюю стороны многоугольника, служит процедура glPolygonMode(GLenum face, GLenum mode). Параметр face может принимать значения GL_FRONT_AND_BACK (обе стороны), GL_FRONT (лицевая сторона) или GL_BACK (нелицевая сторона). Параметр mode может принимать значения GL_POINT, GL_LINE или GL_FILL, обозначая, что многоугольник должен рисоваться как набор граничных точек, граничная ломаная линия или заполненная область, например:

glPolygonMode(GL_FRONT, GL_FILL);

glPolygonMode(GL_BACK, GL_LINE).

По умолчанию вершины многоугольника, которые появляются на экране в направлении против часовой стрелки, называются лицевыми. Это можно изменить при помощи процедуры glFrontFace(GLenum mode). По умолчанию параметр mode равняется GL_CCW, что соответствует направлению обхода против часовой стрелки. Если задать этот параметр равным GL_CW, то лицевыми будут считаться многоугольники с направлением обхода вершин по часовой стрелке.

При помощи процедуры glCullFace(GLenum mode) вывод лицевых или нелицевых граней многоугольников можно запретить. Параметр mode принимает одно из значений GL_FRONT (оставить только лицевые грани), GL_BACK (оставить нелицевые) или GL_FRONT_AND_BACK (оставить все грани). Для отсечения граней необходимо разрешить отсечение при помощи команды glEnable(GL_CULL_FACE).

Шаблон для заполнения грани можно задать при помощи процедуры gIPolygonStipple(const GLubyte * mask), где mask задает массив битов размером 32 на 32.

Для разрешения использования шаблонов при выводе многоугольников служит команда glEnable(GL_POLYGON_STIPPLE).

Вектор нормали для каждой вершины можно задать при помощи одной из следующих процедур:

glNormal3{b s i d f}(TYPE nx, TYPE ny, TYPE nz);

glNormal3{b s I d f}v(const TYPE * v).

В версиях с суффиксами b, s или i значения аргументов масштабируются в отрезок [-1, 1].

В качестве примера приведем процедуру, строящую прямоугольный параллелепипед с ребрами, параллельными координатным осям, по диапазонам изменения x, y и z.

```
#include <windows.h>
#include <gl\gl.h>
drawBox(GLfloat x1, GLfloat x2, GLfloat y1, GLfloat y2, GLfloat z1, GLfloat
z2)
```

```
{
glBegin ( GL_POLYGON );
glNormal3f ( 0.0, 0.0, 1.0 );
glVertex3f ( x1, y1, z2 );
glVertex3f ( x2, y1, z2 );
glVertex3f ( x2, y2, z2 );
glVertex3f ( x1, y2, z2 );
glEnd ();

glBegin ( GL_POLYGON );
glNormal3f ( 0.0, 0.0, -1.0 );
glVertex3f ( x2, y1, z1 );
glVertex3f ( x1, y1, z1 );
glVertex3f ( x1, y2, z1 );
glVertex3f ( x2, y2, z1 );
glEnd ();

glBegin ( GL_POLYGON );
glNormal3f ( -1.0, 0.0, 0.0 );
glVertex3f ( x1, y1, z1 );
glVertex3f ( x1, y1, z2 );
glVertex3f ( x1, y2, z2 );
glVertex3f ( x1, y2, z1 );
glEnd ();

glBegin ( GL_POLYGON );
glNormal3f ( 1.0, 0.0, 0.0 );
glVertex3f ( x2, y1, z2 );
glVertex3f ( x2, y1, z1 );
glVertex3f ( x2, y2, z1 );
glVertex3f ( x2, y2, z2 );
glEnd ();

glBegin ( GL_POLYGON );
glNormal3f ( 0.0, 1.0, 0.0 );
glVertex3f ( x1, y2, z2 );
glVertex3f ( x2, y2, z2 );
glVertex3f ( x2, y2, z1 );
glVertex3f ( x1, y2, z1 );
glEnd ();
```

```
glBegin ( GL_POLYGON );
glNormal3f ( 0.0, -1.0, 0.0 );
glVertex3f ( x2, y1, z2 );
glVertex3f ( x1, y1, z2 );
glVertex3f ( x1, y1, z1 );
glVertex3f ( x2, y1, z1 );
glEnd ();
}
```

Трехмерные фигуры

Функции для построения сплошных 3D фигур:

- auxSolidSphere(R) // сфера
- auxSolidCube(width) // куб
- auxSolidBox(width, height, depth) // коробка
- auxSolidTorus(r,R) // тор
- auxSolidCylinder(r,height) // цилиндр
- auxSolidCone(r,height) // конус
- auxSolidIcosahedron(width) // многогранники
- auxSolidOctahedron(width)
- auxSolidTetrahedron(width)
- auxSolidDodecahedron(width)
- auxSolidTeapot(width) // рисует чайник

Для построения каркасных фигур вместо Solid необходимо использовать Wire. Пример:

auxWireCube(1) // рисует каркасную модель куба.

Преобразование объектов в пространстве

В процессе построения изображения координаты вершин подвергаются определенным преобразованиям. Подобным преобразованиям подвергаются заданные векторы нормали.

Изначально камера находится в начале координат и направлена вдоль отрицательного направления оси Oz.

В OpenGL существуют две матрицы, последовательно применяющиеся в преобразовании координат. Одна из них – матрица моделирования (modelview matrix), а другая – матрица проецирования (projection matrix). Первая служит для задания положения объекта и его ориентации, вторая отвечает за выбранный способ проецирования. OpenGL поддерживает два типа проецирования – параллельное и перспективное.

Существует набор различных процедур, умножающих текущую матрицу (моделирования или проецирования) на матрицу выбранного геометрического преобразования.

Текущая матрица задается при помощи процедуры glMatrixMode(GLenum mode). Параметр mode может принимать значения GL_MODELVIEW, GL_TEXTURE или GL_PROJECTION, позволяя выбирать в качестве текущей матрицы матрицу моделирования (видовую матрицу), матрицу проецирования или матрицу преобразования текстуры.

Процедура glLoadldenity() устанавливает единичную текущую матрицу. Обычно задание соответствующей матрицы начинается с установки единичной матрицы и последовательного применения матриц геометрических преобразований.

Преобразование переноса задается процедурой glTranslate{f d}(TYPE x, TYPE y, TYPE z), обеспечивающей перенос объекта на величину (x, y, z).

Преобразование поворота задаётся процедурой glRotate{f d}(TYPE angle, TYPE x, TYPE y, TYPE z), обеспечивающей поворот на угол angle в направлении против часовой стрелки вокруг прямой с направляющим вектором (x, y, z).

Преобразование масштабирования задаётся процедурой glScale{f d}(TYPE x, TYPE y, TYPE z).

Если указано несколько преобразований, то текущая матрица в результате будет последовательно умножена на соответствующие матрицы.

Получение проекций

Видимым объемом при перспективном преобразовании в OpenGL является усеченная пирамида.

Для задания перспективного преобразования в OpenGL служит процедура:

gIFrustrum(GLdouble teft, GLdoubte right, GLdouble bottom, GLdouble top, GLdouble near, GLdoubte far).

Параметры определяют плоскости, по которым проводится отсечение. Величины near и far должны быть неотрицательными.

Иногда для задания перспективного преобразования удобнее воспользоваться следующей процедурой из библиотеки утилит OpenGL:

gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar).

Эта процедура создает матрицу для задания симметричного поля зрения и умножает текущую матрицу на неё. Здесь fovy – угол зрения камеры в плоскости Oxz, лежащий в диапазоне [0, 180]. Параметр aspect – отношение ширины области к её высоте, zNear и zFar – расстояния вдоль отрицательного направления оси Oz, определяющие ближнюю и дальнюю плоскости отсечения.

Существует ещё одна удобная функция для задания перспективного проецирования:

gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
GIdouble centerX, GLdouble centerY, GLdouble centerZ,
GLdouble upX, GLdouble upY, GLdouble up2).

Вектор (eyeX, eyeY, eyeZ) задаёт положение наблюдателя, вектор (cenlerX, centerY, centerZ) – направление на центр сцены, а вектор (upX, upY, upZ) – направление вверх.

В случае параллельного проецирования видимым объемом является прямоугольный параллелепипед. Для задания параллельного проецирования служит процедура:

glOrtho(GLdouble left, GLdouble right, GLdouble
bottom, GLdouble top, GLdouble near, GLdouble far).

Параметры left и right определяют координаты левой и правой вертикальных плоскостей отсечения, а bottom и top – нижней и верхней горизонтальных плоскостей.

Следующим шагом в задании проецирования (после выбора параллельного или перспективного преобразования) является задание области в окне, в которую будет помещено получаемое изображение. Для этого служит процедура:

gIViewport(GLint x, GLint y, GLsizei width, GLsizei height).

Здесь (x ,y) задаёт нижний левый угол прямоугольной области в окне, а width и height являются её шириной и высотой.

OpenGL содержит стек матриц для каждого из трёх типов преобразований. При этом текущую матрицу можно поместить в стек или извлечь матрицу из стека и сделать её текущей.

Для помещения текущей матрицы в стек служит процедура gIPushMatrix(), для извлечения матрицы из стека – процедура glPopMatrix().

Задание моделей закрашивания

Линия или заполненная грань могут быть нарисованы одним цветом (плоское закрашивание, GL_FLAT) или путём интерполяции цветов в вершинах (закрашивание Гуро, GL_SMOOTH).

Для задания режима закрашивания служит процедура glShadeModel(GLenum mode), где параметр mode принимает значение GL_SMOOTH или GL_FLAT.

Освещение

OpenGL использует модель освещённости, в которой свет приходит из нескольких источников, каждый из которых может быть включён или выключен. Кроме того, существует еще общее фоновое (ambient) освещение.

Для правильного освещения объектов необходимо для каждой грани задать материал, обладающий определенными свойствами. Материал может испускать свой собственный свет, рассеивать падающий свет во всех

направлениях (диффузное отражение) или подобно зеркалу отражать свет в определенных направлениях.

Пользователь может определить до восьми источников света и их свойства, такие, как цвет, положение и направление. Для задания этих свойств служит процедура:

gllight{i f}[v](GLenum light, GLenum pname, TYPE param),

которая задаёт параметры для источника света light, принимающего значения GL_LIGHT0, GL_LIGHT1, ..., GL_LIGHT7. Параметр pname определяет характеристику источника света, которая задается последним параметром.

Для использования источников света расчёт освещенности следует разрешить командой glEnable(GL_LGHTING), а применение соответствующего источника света разрешить (включить) командой glEnable, например: glEnable(GL_LIGHT0).

Источник света можно рассматривать как имеющий вполне определенные координаты и светящий во всех направлениях или как направленный источник, находящийся в бесконечно удаленной точке и светящий в заданном направлении (x, y, z).

Если параметр w в команде GL_POSITION равен нулю, то соответствующий источник света – направленный и светит в направлении (x, y, z). Если же w отлично от нуля, то это позиционный источник света, находящийся в точке с координатами (x/w, y/w, z/w).

Заданием параметров GL_SPOT_CUTOFF и GL_SPOT_DIRECTION можно создавать источники света, которые будут иметь коническую направленность. По умолчанию значение параметра GL_SPOT_CUTOFF равно 180°, т. е. источник светит во всех направлениях с равной интенсивностью. Параметр GL_SPOT_CUTOFF определяет максимальный угол от направления источника, котором распространяется свет от него. Он может принимать значение 180° (не конический источник) или от 0 до 90°.

Интенсивность источника с расстоянием – убывает (параметры этого убывания задаются при помощи параметров GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION и GL_QUADRATIC_ATTENUATION). Только собственное свечение материала и глобальная фоновая освещенность с расстоянием не ослабевают.

Глобальное фоновое освещение можно задать при помощи команды

gtLightModel{i f} [v] (GL_LIGHT_MODEL_AMBIENT, ambientColor).

Местонахождение наблюдателя оказывает влияние на блики на объектах. По умолчанию при расчётах освещённости считается, что наблюдатель находится в бесконечно удалённой точке, т. е. направление источника света на наблюдателя постоянно для любой вершины. Можно включить более реалистическое освещение, когда направление источника света на наблюдателя будет вычисляться для каждой вершины отдельно. Для этого служит команда:

glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE).

Для задания освещения как лицевых, так и нелицевых граней (для нелицевых граней вектор нормали переворачивается) служит следующая команда:

glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE).

Причём существует возможность отдельного задания свойств материала для каждой из сторон.

Свойства материала, из которого сделан объект, задаются при помощи процедуры:

glMaterial{i f}[v](GLenum face, GLenum pname, TYPE param).

Параметр face указывает, для какой из сторон грани задается свойство, и принимает одно из следующих значений: GL_BACK, GL_FRONT_AND_BACK, GL_FRONT.

Параметр pname указывает, какое именно свойство материала задается. Расчёт освещённости в OpenGL не учитывает затенения одних объектов другими.

Полупрозрачность

Полупрозрачность объектов устанавливается с использованием α-канала (в RGBA-представлении цвета). Если его значение равно единице – объект непрозрачен. Задавая значения, отличные от единицы, можно смешивать цвет выводимого пикселя с цветом пикселя, уже находящегося в соответствующем месте на экране, создавая тем самым эффект прозрачности.

α-значение характеризует степень поглощения фрагментом проходящего через него света. Так, если у стекла установить значение, равное 0,2, то в результате вывода цвет получившегося фрагмента будет на 20 % состоять из собственного цвета стекла и на 80 % – из цвета фрагмента под ним.

Для использования α-канала необходимо сначала разрешить режим прозрачности и смешения цветов командой gEnable(GL_BLEND).

В процессе смешения цветов цветовые компоненты выводимого фрагмента $R_sG_sB_sA_s$ смешиваются с цветовыми компонентами уже выведенного фрагмента $R_dG_dB_dA_d$ по формуле

$$(R_sS_r+R_dD_r,\ G_sS_g+G_dD_g,\ B_sS_b+B_dD_b,\ A_sS_a+A_dD_a),$$

где (S_r, S_g, S_b, S_a) и (D_r, D_g, D_b, D_a) – коэффициенты смешения.

Для задания связи этих коэффициентов с α-значениями используется следующая функция:

glBlendFunc(GLenum sfactor, GLenum dfactor).

Здесь параметр sfactor задаёт то, как нужно вычислять коэффииенты (S_r, S_g, S_b, S_a), а параметр dfactor – коэффициенты (D_r, D_g, D_b, D_a).

Наложение текстуры

Текстурирование позволяет наложить изображение на многоугольник и вывести этот многоугольник с наложенной на него текстурой, соответствующим образом преобразованной. OpenGL поддерживает одно- и двумерные текстуры, а также различные способы наложения текстуры.

Для использования текстуры надо сначала разрешить одно- или двумерное текстурирование при помощи команд glEnable(GL_TEXTURE1D) или glEnable(GL_TEXTURE_2D).

Для задания двумерной текстуры служит процедура:

glTexlmage2D(GLenum target, GLint level, GLint
component, GLsizei width, GLsizei height, GLint border,
GLenum format, GLenum type, const GLvoid *pixels).

Параметр target зарезервирован для будущего использования и в текущей версии OpenGL должен быть равен GL_TEXTURE_2D. Параметр level используется в том случае, если задается несколько разрешений данной текстуры. При ровно одном разрешении он должен быть равным нулю.

Следующий параметр – component – целое число от 1 до 4, показывающее, какие из RGBA-компонентов выбраны для использования. Значение 1 выбирает компонент R, значение 2 выбирает R и A компоненты, 3 соответствует R, G и B, а 4 соответствует компонентам RGBA.

Параметры width и height задают размеры текстуры, border задает размер границы (бортика), обычно равный нулю. Как параметр width, так и параметр height, должны иметь вид $2^n + 2b$, где n – целое число, а b – значение параметра border. Максимальный размер текстуры зависит от реализации OpenGL, но он не менее 64 на 64.

При текстурировании OpenGL поддерживает использование пирамидального фильтрования (mip-mappping). Для этого необходимо иметь текстуры всех промежуточных размеров, являющихся степенями двух, вплоть до 1x1, и для каждого такого разрешения вызвать glTexImage2D с соответствующими параметрами level, width, height и image. Кроме того, необходимо задать способ фильтрования, который будет применяться при выводе текстуры.

Под фильтрованием здесь подразумевается способ, которым для каждого пикселя будет выбираться подходящий элемент текстуры (тексель). При текстурировании возможна ситуация, когда 1 пикселю соответствует небольшой фрагмент текселя (увеличение) или же, наоборот, когда 1 пикселю соответствует целая группа текселей (уменьшение).

Способ выбора соответствующего текселя как для увеличения, так и для уменьшения (сжатия) текстуры необходимо задать отдельно. Для этого используется процедура:

glTexParameteri(GL_TEXTURE_2D, GLenum p1, GLenum p2),

где параметр p1 показывает, задается ли фильтр для сжатия или для растяжения

текстуры, принимая значение GL_TEXTURE_MIN_FLITER или GL_TEXTURE_MAG_FILTER. Параметр p2 задает способ фильтрования.

При использовании пирамидального фильтрования помимо выбора текселя на одном слое текстуры появляется возможность либо выбрать один соответствующий слой, либо проинтерполировать результаты выбора между двумя соседними слоями. Для правильного применения текстуры каждой вершине следует задать соответствующие ей координаты текстуры при помощи процедуры:

glTexCoord{1 2 3 4}{s i f d}[v](TYPE coord, ...).

Этот вызов задаёт значения индексов текстуры для последующей команды glVertex.

Если размер грани больше, чем размер текстуры, то для циклического повторения текстуры служат команды:

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_S_WRAP, GL_REPEAT);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_T_WRAP, GL_REPEAT).

Координаты текстуры обычно подвергаются преобразованию при помощи матрицы текстурирования. По умолчанию она совпадает с единичной матрицей, но пользователь сам имеет возможность задать преобразования текстуры, например, следующим образом:

glMatrixMode(GL_TEXTURE);

glRotatef(...};

glMatrixMode(GL_MODELV1EW).

При выводе текстуры OpenGL может использовать линейную интерполяцию или точно учитывать перспективное искажение. Для задания точного текстурирования служит команда:

glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST).

Если качество не играет большой роли, а нужна высокая скорость рендеринга, то в качестве последнего аргумента следует использовать константу GL_FASTEST.

Описанный выше способ работ с текстурами используется в OpenGL версии 1.0. В более новых версиях OpenGL, начиная с версии 1.1, введены дополнительные функции, повышающие удобство работы с текстурами. В OpenGL 1.0 процедуру glTexImage2D необходимо вызывать всякий раз, когда нужно сменить текущую текстуру. Это достаточно медленный способ. В OpenGL 1.1 имеется возможность присваивать имена текстурам и затем сменять текущую текстуру только указанием имени новой текстуры, без повторной её загрузки в память процедурой glTexImage2D.

Имя текстуры представляет собой уникальное значение типа GLuint. Перед использованием текстуры необходимо присвоить ей имя. Имена текстур можно сгенерировать при помощи процедуры

glGenTextures(GLsizei n, GLuint *textures).

Параметр n определяет количество текстур, для которых необходимо сгенерировать имена. Параметр textures является указателем на массив переменных типа GLuint, состоящим из n элементов. После вызова процедуры каждый элемент массива будет содержать уникальное имя текстуры, которое затем может быть использовано при работе.

Для выбора текущей (активной) текстуры используется функция:

glBindTexture(GLenum target, GLuint texture).

Параметр target определяет тип текстуры (одномерная – GL_TEXTURE_1D или двумерная – GL_TEXTURE_2D). На практике более часто используются двумерные текстуры, которые представляют собой обычные двумерные изображения. Параметр texture определяет имя текстуры, которую необходимо сделать активной.

После того, как установлена активная текстура, можно вызвать процедуру glTexImage2D и задать параметры текстуры, а также сами её тексели. После вызова процедуры glTexImage2D текстура готова к применению.

Для того чтобы наложить текстуру на объект или многоугольник достаточно установить активную текстуру (процедура glBindTexture) и определить текстурные координаты при помощи процедуры glTexCoord.

Достоинство использования функций OpenGL 1.1 для работы с текстурами заключается не только в более высоком быстродействии по сравнению с использованием процедуры glTexImage2D, но и в повышенном удобстве работы с текстурами. Создав массив текстурных имён можно работать одновременно с несколькими текстурами, вызывая лишь функцию glBindTexture по мере необходимости.

При написании программы, которая покрывает куб текстурой, необходимо учитывать то, что саму текстуру требуется загружать из отдельного файла.

Для загрузки текстуры, а также задания ее параметров рекомендуется написать отдельную функцию. Например, если текстура размером 64x64, то считывание ее в массив выглядит так:

```
Bits: Array [0..63, 0..63, 0..2] of GLubyte;
For i := 0 to 63 do
    For j := 0 to 63 do begin
      bits [i, j, 0] := GetRValue(bitmap.Canvas.Pixels[i,j]);
      bits [i, j, 1] := GetGValue(bitmap.Canvas.Pixels[i,j]);
      bits [i, j, 2] := GetBValue(bitmap.Canvas.Pixels[i,j]);
end.
```

Последний компонент трехмерного массива – это цветовые составляющие пиксела RGB.

Загрузить текстуру в память можно следующей командой:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 64, 64, 0, GL_RGB,
GL_UNSIGNED_BYTE, @Bits);
```

После этого необходимо разрешить работать с текстурой:

```
glEnable(GL_TEXTURE_2D);
```

Следующий этап – это привязка текстурных координат к координатам объектов. В обработчике создания окна пишем (для одной грани куба):

```
    glBegin (GL_QUADS);
    glTexCoord2d (1.0, 0.0);
    glVertex3f (-1.0, -1.0, 1.0);
    glTexCoord2d (1.0, 1.0);
    glVertex3f (1.0, -1.0, 1.0);
    glTexCoord2d (0.0, 1.0);
    glVertex3f (1.0, -1.0, -1.0);
    glTexCoord2d (0.0, 0.0);
    glVertex3f (-1.0, -1.0, -1.0);
glEnd.
```

ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЗАДАНИЙ

Лабораторные работы необходимо выполнять с использованием языка Visual C++. Или по согласованию с преподавателем на Builder C++, Delphi.

На первые работы приведены листинги программ. Последующие работы планируется выполнять самостоятельно.

При выполнении работ необходимо оформлять отчет, который должен быть представлен в печатном и электронном виде. Далее студент проходит собеседование.

Печатный отчет студента по заданию должен содержать:

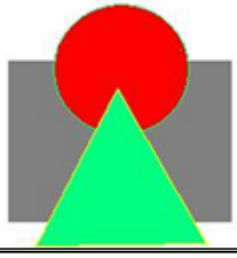
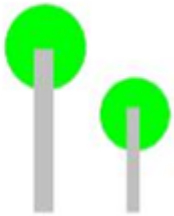
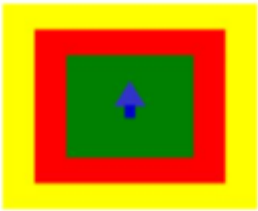
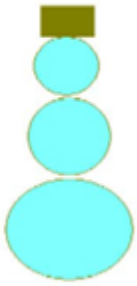
1. Титульный лист.
2. Лист с формулировкой задания.
3. Листы с текстом программы (листингом). В котором сложные или ключевые моменты должны быть прокомментированы.
4. В отчет необходимо вставлять рисунки, содержащие результаты выполнения программ.
5. Заключительный лист может содержать рекомендации по улучшению программы.

Электронный отчет может находиться на любом носителе информации (дискета, компакт-диск и др.) и должен включать отчет, исполняемый файл программы и ее исходный текст.

Задание 1

1. Изучить теоретические основы OpenGL, прил. 1 и 2.
2. Построить прямоугольник. Листинг программы приведен в прил. 1. Изменить параметры освещенности, цвета объекта и фона, размера.
3. В соответствии со своим вариантом (табл. 2) построить 2D объекты.

Таблица 2 - Исходные задания к заданию 1

Номер варианта	Задание	Номер варианта	Задание
1		6	
2		7	
3		8	
4		9	
5		10	

Задание 2

1. Ознакомиться с теоретическими основами и прил. 3–8.
2. Построить каркасные и сплошные модели фигур и поверхностей, (прил. 3). Изменить цвет объектов.
3. Выполнить задание в соответствии со своим вариантом (табл. 3) (по согласованию с преподавателем задание может быть изменено). Ввести различное освещение объектов.

Таблица 3 - Исходные задания к заданию 2

Номер варианта	Задание	Номер варианта	Задание
1	Собака	6	Ель
2	Олимпийские кольца	7	Марсианин
3	Цветы	8	Медведь
4	Ваза	9	Снеговик
5	Стол с чайником	10	Пингвин

Задание 3

1. Ознакомиться с теоретическими основами и прил. 5, 9.
2. На объекты, реализованные в задании 2 нанести текстуры (по согласованию с преподавателем задание может быть изменено).

Задание

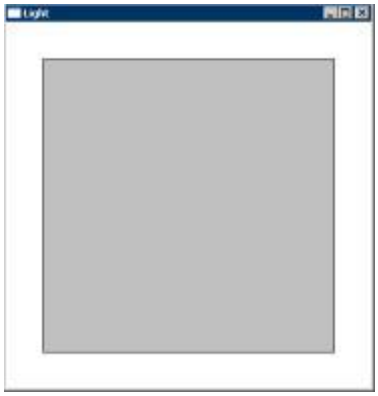
1. Ознакомиться с прил. 9–12.
2. Анимировать результаты выполненного задания 3(по согласованию с преподавателем задание может быть изменено).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Построение квадрата

Результат выполнения программы



Текст программы

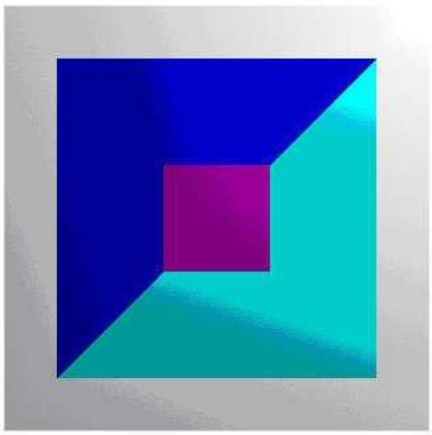
```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glColor3d(1,1,1);
 glBegin(GL_QUADS);
  glVertex3d(-4,-4,0);
  glVertex3d(-4, 4,0);
  glVertex3d( 4, 4,0);
  glVertex3d( 4,-4,0);
 glEnd();
/*
 glColor3d(1,0,0);
```

```
 auxSolidSphere(0.1);
*/
  auxSwapBuffers();
}
void main()
{
float pos[4] = {3,3,3,0.5};
float dir[3] = {-1,-1,-1};
    GLfloat mat_specular[] = {1,1,1,1};
    auxInitPosition( 50, 10, 400, 400);

    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Light" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);

    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
   glLighti(GL_LIGHT0, GL_SPOT_EXPONENT, 0);
   glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 90);
auxMainLoop(display);
}
```

ПРИЛОЖЕНИЕ 2
Построение 2D объектов
Результат выполнения программы



Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glColor3d(1,1,1);
 glBegin(GL_QUADS);
  glVertex3d(-4,-4,0);
  glVertex3d(-4, 4,0);
  glVertex3d( 4, 4,0);
  glVertex3d( 4,-4,0);
 glEnd();
 glColor3d(0,1,1);
 glBegin(GL_QUADS);
```

```
  glVertex3d(-3,-3,1);
  glVertex3d(-3, 3,1);
  glVertex3d( 3, 3,1);
  glVertex3d( 3,-3,1);
 glEnd();
  glColor3d(0,0,1);
 glBegin(GL_TRIANGLES);
  glVertex3d(-3,-3,2);
  glVertex3d(-3, 3,2);
  glVertex3d( 3, 3,2);
  //glVertex3d( 3,-3,1);
 glEnd();
 glColor3d(1,0,1);
 glBegin(GL_QUADS);
  glVertex3d(-1,-1,3);
  glVertex3d(-1, 1,3);
  glVertex3d( 1, 1,3);
  glVertex3d( 1,-1,3);
 glEnd();
/*
 glColor3d(1,0,0);
 auxSolidSphere(0.1);
*/
  auxSwapBuffers();
}
void main()
{
float pos[4] = {3,3,3,0.5};
float dir[3] = {-1,-1,-1};
   GLfloat mat_specular[] = {1,1,1,1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Light" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

```
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
   glLighti(GL_LIGHT0, GL_SPOT_EXPONENT, 0);
   glLighti(GL_LIGHT0, GL_SPOT_CUTOFF, 90);
   auxMainLoop(display);
}
```

ПРИЛОЖЕНИЕ 3

Построение 3D объектов

Результат выполнения программы

Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
 glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glPushMatrix();
glTranslated(0.5,4,0);
glColor3d(0,0,1);
auxSolidCube(1);      // куб
glTranslated(0,-2,0);
glColor3d(0,1,0);
auxSolidBox(1,0.75,0.5); // коробка
glTranslated(0,-2,0);
glColor3d(0,1,1);
```

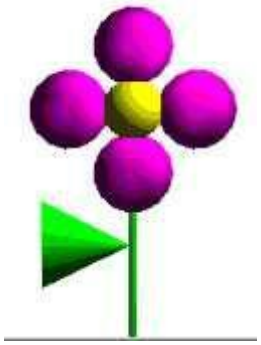
```
auxSolidTorus(0.2,0.5); // тор
glTranslated(0,-2,0);
glColor3d(1,0,0);
auxSolidCylinder(0.5,1); // цилиндр
glTranslated(0,-2,0);
glColor3d(0,1,0);
auxSolidCone(1,1);   // конус
glTranslated(2,8,0);
glColor3d(1,0,1);
auxSolidIcosahedron(1);
// многогранники
glTranslated(0,-2,0);
glColor3d(1,1,1);
auxSolidOctahedron(1);
glTranslated(0,-2,0);
glColor3d(0,1,1);
auxSolidTeapot(0.7); // чайник
glTranslated(0,-2,0);
glColor3d(0,1,0);
 auxSolidTetrahedron(1);
 glTranslated(0,-2,0);
 glColor3d(1,1,0);
 auxSolidDodecahedron(1);
 glTranslated(-6,8,0);
 glColor3d(0,0,1);
 auxWireCube(1);    // каркасная модель куба
 glTranslated(0,-2,0);
  glColor3d(0,1,0);
 auxWireBox(1,0.75,0.5); // каркасная модель параллелограмма
 glTranslated(0,-2,0);
    glColor3d(0,1,1);
  auxWireTorus(0.2,0.5); // каркасная модель тора
  glTranslated(0,-2,0);
        glColor3d(1,0,0);
  auxWireCylinder(0.5,1); // каркасная модель цилиндра
   glTranslated(0,-2,0);
        glColor3d(0,1,0);
  auxWireCone(1,1); // каркасная модель конуса
  glTranslated(2,8,0);
        glColor3d(1,0,1);
```

```
   auxWireIcosahedron(1); // каркасные модели многогранников
   glTranslated(0,-2,0);
         glColor3d(1,1,1);
   auxWireOctahedron(1);
   glTranslated(0,-2,0);
         glColor3d(0,1,1);
   auxWireTeapot(0.7); // каркасная модель чайника
   glTranslated(0,-2,0);
         glColor3d(0,1,0);
   auxWireTetrahedron(1);
   glTranslated(0,-2,0);
         glColor3d(1,1,0);
   auxWireDodecahedron(1);
  glPopMatrix();
   auxSwapBuffers();
}
void main()
{
       float pos[4] = {3,3,3,1};
       float dir[3] = {-1,-1,-1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Shapes" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
   auxMainLoop(display);
}
```

ПРИЛОЖЕНИЕ 4

Цветок

Результат выполнения программы



Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
 glClear( GL_COLOR_BUFFER_BIT |
 GL_DEPTH_BUFFER_BIT ); glPushMatrix();
glColor3d(0,1,0);
 auxSolidCylinder(0.1,10);
   glColor3d(1,1,0);
 auxSolidSphere(0.7);
 glColor3d(1,0,1);
 glTranslated(0,-1.4,0);
 auxSolidTorus(0.8,0.1);
  glTranslated(0,2.8,0);
 auxSolidTorus(0.8,0.1);
```

```
 glTranslated(1.4,-1.4,0);
 auxSolidTorus(0.8,0.1);
 glTranslated(-2.8,0,0);
 auxSolidTorus(0.8,0.1);
 glTranslated(-0.6,-3,0);
 glColor3d(0,1,0);
 glRotated(90,0,1,0);
 auxSolidCone(1,2);
 glPopMatrix();
  auxSwapBuffers();
}
void main()
{
float pos[4] = {3,3,3,1};
float dir[3] = {-1,-1,-1};
  auxInitPosition( 50, 10, 400, 400);
  auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
  auxInitWindow( "Shapes" );
  auxIdleFunc(display);
  auxReshapeFunc(resize);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_COLOR_MATERIAL);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_POSITION, pos);
  glLightfv(GL_LIGHT0,
  GL_SPOT_DIRECTION, dir);
  auxMainLoop(display);

}
```

ПРИЛОЖЕНИЕ 5
Снеговик с текстурой
Результат выполнения программы

Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
GLuint texture[2];
GLvoid LoadGLTextures()
{
        AUX_RGBImageRec *texture1;
        texture1 = auxDIBImageLoad("c:/Data/Tela1.bmp");

// Создание текстуры
glGenTextures(1, &texture[0]);
glBindTexture(GL_TEXTURE_2D,
texture[0]);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_L
        INEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,G
        L_LINEAR); glTexImage2D(GL_TEXTURE_2D, 0, 3, texture1->sizeX,
        texture1->sizeY, 0,
   GL_RGB, GL_UNSIGNED_BYTE, texture1->data);
}
```

```
void CALLBACK resize(int width,int height)
{
  LoadGLTextures();              // Загрузка текстур
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
   GLUquadricObj *q; // Квадратичный объект для рисования сферы q =
   gluNewQuadric(); // Создать квадратичный объект тип генерируемых
   нормалей для него – «сглаженные»
   gluQuadricNormals(q, GL_SMOOTH);
   // Включить текстурные координаты для объекта
   gluQuadricTexture(q, GL_TRUE);
// Настройка сферического наложения
   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
   // Настройка отображения сферы
   glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
   GL_SPHERE_MAP); glClear( GL_COLOR_BUFFER_BIT |
   GL_DEPTH_BUFFER_BIT );
   glBindTexture(GL_TEXTURE_2D, texture[0]);
  glPushMatrix();
   glEnable(GL_TEXTURE_2D);      // Разрешение наложение текстуры
         gluSphere(q,1.5,32,16);// Рисование первой сферы
         glDisable(GL_TEXTURE_2D);        // Разрешение наложение текстуры
         glTranslated(0,-3,0);
         auxSolidSphere(2.0);
         glTranslated(0,3,0);
         auxSolidSphere(1.5);
         glTranslated(0,2,0);
         auxSolidSphere(1);
         glColor3d(0,0,1);   //Левый глаз
         glTranslated(-0.3,0.3,1);
   auxSolidSphere(0.1);
         glColor3d(0,0,0);      //Правый глаз
```

```
        glTranslated(0.6,0,0);
  auxSolidSphere(0.2);
        glTranslated(-0.3,-0.3,0.1);      //Бандана
        glColor3d(1,0,0);
  auxSolidSphere(0.3);
        glTranslated(0,0.50,-1);//Нос
  glColor3d(0,1,0);
  auxSolidSphere(0.75);
        //glTranslated(0,0.75,-1);
  //glColor3d(0.75,0.75,0.75);
  //auxSolidSphere(0.75);
 glBegin(GL_LINES);      //Повязка
  glColor3d(0,0,0);
  glLineWidth(20);
  glVertex3d(-0.15,0.25,0.69);
        glVertex3d(0.94,-0.8,1);
 glEnd();
 glBegin(GL_TRIANGLES);  //Завязка1
  glColor3d(0,1,0);
  glVertex3d(0.75,0.2,1);
        glVertex3d(1,0.8,1);
        glVertex3d(1.5,0.6,1);
 glEnd();
 glBegin(GL_TRIANGLES);  //Завязка2
  glColor3d(0,1,0);
  glVertex3d(0.75,0.2,1);
        glVertex3d(1.2,-0.2,1);
        glVertex3d(1.5,0.2,1);
 glEnd();
  glColor3d(0,1,0);        //Узелок
        glTranslated(0.78,0.21,1);
  auxSolidSphere(0.1);
  glColor3d(0.75,0.75,0.75);   //Рука правая
        glTranslated(1,-2,0);
  auxSolidSphere(0.55);
  glColor3d(0.75,0.75,0.75);   //Рука левая
        glTranslated(-3.5,-0.12,0);
  auxSolidSphere(0.55);
glPopMatrix();
```

```
   auxSwapBuffers();
}
void main()
{
       float pos[4] = {3,3,3,1};
       float dir[3] = {-1,-1,-1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Snowman" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_ALPHA_TEST);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_BLEND);
   glBlendFunc(GL_SRC_ALPHA,
   GL_ONE_MINUS_SRC_ALPHA);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
         auxMainLoop(display);
}
```

ПРИЛОЖЕНИЕ 6

Новогодняя ель

Результат выполнения программы

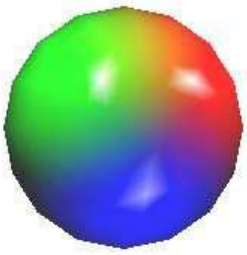
Текст программы

```
#include "stdafx.h"
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
   glClear( GL_COLOR_BUFFER_BIT |
   GL_DEPTH_BUFFER_BIT ); glPushMatrix();
   glColor3d(0,1,0);
   glTranslated(0,-4,0);
   glRotated(90,0,1,0);
   glRotated(270,1,0,0);
   auxSolidCone(4,4);
   glTranslated(0.5,0,-0.5);
```

```
   glColor3d(1,0.4,0.2);
   auxSolidCylinder(0.5,2);
   glTranslated(0,0,0.5);
   glColor3d(0,1,0);
   glTranslated(0,0,2);
   auxSolidCone(3.5,3.5);
   glTranslated(0,0,2);
   auxSolidCone(3,3);
   glTranslated(0,0,2);
   auxSolidCone(2,2);
   glTranslated(-3.4,-1,-2);
   glColor3d(1,0,0);
   auxSolidSphere(0.5);
   glTranslated(-0.12,-1,-2);
   glColor3d(0,1,1);
   auxSolidSphere(0.7);
   glTranslated(-0.23,3,3);
   glColor3d(1,1,0);
   auxSolidSphere(0.5);
   glTranslated(0,0,-3);
   glColor3d(1,0,1);
   auxSolidSphere(0.5);
   glPopMatrix();
   auxSwapBuffers();
}
int main(int argc, char* argv[])
{
   float pos[4] = {3,3,0.24,1};
   float dir[3] = {1,1,1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH |
   AUX_DOUBLE ); auxInitWindow( "С Новым Годом!!!" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_ALPHA_TEST);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
```

```
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA,
    GL_ONE_MINUS_SRC_ALPHA);
    glLightfv(GL_LIGHT0, GL_POSITION, pos);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
    auxMainLoop(display);
        return 0;
}
```

ПРИЛОЖЕНИЕ 7
Работа с источниками освещения
Результат выполнения программы



Текст программы

```
#include <windows.h>
#include <math.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
float color1[4] = {1,0,0,1};
float color2[4] = {0,1,0,1};
float color3[4] = {0,0,1,1};
int time=0;
void CALLBACK resize(int width,int height)
{
    glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, -5,5);
  gluLookAt( 4*cos(time),0,4*sin(time), 0,0,0,
  0,1,0 ); glMatrixMode( GL_MODELVIEW );
  time++;
}
void CALLBACK display(void)
{
 float l1,l2,l3;
 glClear( GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT ); glColor3d(1,1,1);
      auxSolidSphere(2);
    //glRotated(2,1,0,1);
  auxSwapBuffers();
```

```
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, -5,5);
  gluLookAt( 4*cos(time/10),0,4*sin(time/10), 0,0,0,
  0,1,0 ); glMatrixMode( GL_MODELVIEW );
  time++;
}
void main()
{
float pos[4] = {3,3,3,1};
float sp[4] = {1,1,1,1};
float mat_specular[] = {1,1,1,1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH |
   AUX_DOUBLE ); auxInitWindow( "Glaux Template" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
    glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT3);
    glEnable(GL_LIGHT5);
    glEnable(GL_LIGHT6);
   glLightfv(GL_LIGHT3, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT5, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT6, GL_SPECULAR, sp);
   //color[1]=color[2]=0;
   glLightfv(GL_LIGHT3, GL_DIFFUSE, color1);
   //color[0]=0; color[1]=1;
   glLightfv(GL_LIGHT5, GL_DIFFUSE, color2);
   //color[1]=0; color[2]=1;
   glLightfv(GL_LIGHT6, GL_DIFFUSE, color3);
   glLightfv(GL_LIGHT3, GL_POSITION, pos);
    pos[0] = -3;
   glLightfv(GL_LIGHT5, GL_POSITION,
    pos); pos[0]=0;pos[1]=-3;

   glLightfv(GL_LIGHT6, GL_POSITION, pos);
   glMaterialfv(GL_FRONT, GL_SPECULAR,
```

```
    mat_specular); glMaterialf(GL_FRONT,
    GL_SHININESS, 128.0); auxMainLoop(display);

}
```

ПРИЛОЖЕНИЕ 8

Текстурирование и анимация снеговика

Результат выполнения программы

Текст программы

```
#include "stdafx.h"
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
AUX_RGBImageRec* photo_image;
unsigned int photo_tex;
void CALLBACK resize(int width,int height)
{glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -5,5, 2,12);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
GLUquadricObj *quadObj;
   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
        quadObj = gluNewQuadric();
        gluQuadricTexture(quadObj, GL_TRUE);
gluQuadricDrawStyle(quadObj, GLU_FILL);
glColor3d(1,1,1);
glRotated(0.5, 0,1,0);
glPushMatrix();
glTranslated(-0.8,-3,0);
 glRotated(-90, 1,0,0);
```

```
  gluSphere(quadObj,1.3, 16, 16);
glPopMatrix();
gluDeleteQuadric(quadObj);
        glPushMatrix ();
        glTranslated(-0.8,-3,0);
   glColor3d(1,1,1);
   //auxSolidSphere(1.3);
        glTranslated(0,2.1,0);
   glColor3d(1,1,1);
   auxSolidSphere(1);
        glTranslated(0,1.5,0); //голова
   glColor3d(1,1,1);
   auxSolidSphere(0.7);
        glTranslated(-0.25,0.35,0.6); //первый глаз
   glColor3d(0,0,1);
   auxSolidSphere(0.1);
        glTranslated(0.5,0,0); //второй глаз
   glColor3d(0,0,1);
   auxSolidSphere(0.1);
        glTranslated(0.9,-1.4,-0.6); //первая рука
   glColor3d(1,1,1);
   auxSolidSphere(0.4);
        glTranslated(-2.2,0,0); //вторая рука
   glColor3d(1,1,1);
   auxSolidSphere(0.4);
        glTranslated(1.05,1.2,0); //нос
   glColor3d(1,0,0);
   auxSolidCone(0.2,2);
   glTranslated(0,0.4,0);
        glRotated(-90,1,0,0);
   glColor3d(0,0,1);
   auxSolidCone(0.5,2);
        glPopMatrix();
        auxSwapBuffers();
}
void main()
{
float pos[4] = {3,3,3,1};
float color[4] = {1,1,1,1};
```

```
float sp[4] = {1,1,1,1};
float mat_specular[] = {1,1,1,1};
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Lab Work OpenGL" ); //заголовок окна
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_LIGHT1);
   glEnable(GL_LIGHT2);
   glLightfv(GL_LIGHT0, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT1, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT2, GL_SPECULAR, sp);
   color[1]=color[2]=0;
   glLightfv(GL_LIGHT0, GL_DIFFUSE, color);
   color[0]=0;color[1]=1;
   glLightfv(GL_LIGHT1, GL_DIFFUSE, color);
   color[1]=0;color[2]=1;
   glLightfv(GL_LIGHT2, GL_DIFFUSE, color);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
     pos[0] = -3;
   glLightfv(GL_LIGHT1, GL_POSITION, pos);
      pos[0]=0;pos[1]=-3;
   glLightfv(GL_LIGHT2, GL_POSITION, pos);
         glEnable(GL_TEXTURE_2D);
   photo_image = auxDIBImageLoad("texture.bmp"); glGenTextures(1,
         &photo_tex);
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
         glBindTexture(GL_TEXTURE_2D,photo_tex);
   gluBuild2DMipmaps(GL_TEXTURE_2D, 3,
               photo_image->sizeX,
               photo_image->sizeY,
               GL_RGB, GL_UNSIGNED_BYTE,
               photo_image->data);
auxMainLoop(display);
}
```

ПРИЛОЖЕНИЕ 9

Вентилятор с вращающимися лопастями

Результат выполнения программы

Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-5,5, -1,9, -10,10);
  gluLookAt( 2,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
        static int angle=0;
        const speed=5;
GLdouble equation[4] = {0,-1,0,0.5};
GLdouble equation1[4] = {0,1,0,0.5};
GLdouble equation2[4] = {-1,0,0,0.5};
GLdouble equation3[4] = {1,0,0,0.5};
 glClear( GL_COLOR_BUFFER_BIT |
 GL_DEPTH_BUFFER_BIT ); // Podstavka
 glEnable(GL_CLIP_PLANE0);
 glClipPlane(GL_CLIP_PLANE0, equation);
 glEnable(GL_CLIP_PLANE1);
 glClipPlane(GL_CLIP_PLANE1, equation1);
```

```
   glColor3d(1,0.75,0.75);
   auxSolidSphere( 3 );
 glDisable(GL_CLIP_PLANE0);
 glDisable(GL_CLIP_PLANE1);
 //nozka
 glPushMatrix();
        glTranslated(0,2.5,0);
        auxSolidCylinder(0.5,4);
 glPopMatrix();
 //korpus;
glPushMatrix();
        glTranslated(0,4,0);
        glRotated(90,0,0,1);
        glTranslated(0,1,0);
        auxSolidCylinder(1,5);
        glTranslated(0,-4.1,0);
        equation[3]=0;
        glEnable(GL_CLIP_PLANE0);
        glClipPlane(GL_CLIP_PLANE0, equation);
        auxSolidSphere(0.8);
        glDisable(GL_CLIP_PLANE0);
        auxSolidCylinder(0.5,1);
        glTranslated(0,-0.3,0);
        glPushMatrix();
                glRotated(-angle*speed,0,1,0);
                glTranslated(0,0,1.5);
                glRotated(30,0,0,1);
                auxSolidBox(0.1,0.5,2);
        glPopMatrix();
        glPushMatrix();
                glRotated(-angle*speed+120,0,1,0);
                glTranslated(0,0,1.5);
                glRotated(30,0,0,1);
                auxSolidBox(0.1,0.5,2);
        glPopMatrix();
        glPushMatrix();
                glRotated(-angle*speed-120,0,1,0);
                glTranslated(0,0,1.5);
                glRotated(30,0,0,1);
```

```
                auxSolidBox(0.1,0.5,2);
        glPopMatrix();
 glPopMatrix();
 angle++;
 auxSwapBuffers();
}
void main()
{
float pos[4] = {3,3,3,1};
float dir[3] = {-1,-1,-1};
   GLfloat mat_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f };
   GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
   auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "ClipPlane" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
   glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
   glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialf(GL_FRONT, GL_SHININESS, 50.0);
   auxMainLoop(display);

}
```

ПРИЛОЖЕНИЕ 10
Анимация – чайник
Результат выполнения программы

Трехмерный вращающийся чайник, который:

- по событию левой кнопки мыши меняет цвет;
- по событию правой кнопки мыши включает или выключает режим вращения;
- по событию клавиш ВВЕРХ/ВНИЗ увеличивает/замедляет вращение;
- по событию клавиши ПРОБЕЛ меняет оси вращения.

Текст программы

```
#include "stdafx.h"
#include <windows.h>
#include <GL\gl.h>
#include <GL\GLU.h>
#include <GL\GLAUX.h>
int clr_number=2;//порядковой номер текущего цвета bool
flag=true;//включение/выключение вращения
int d=5;//приращение угла поворота int ax=1;//положение оси вращения
void CALLBACK resize(int width, int Height);
void CALLBACK display(void);
void CALLBACK mouse_leftbtn( AUX_EVENTREC* event); //обработчик
левой клавиши мыши
void CALLBACK mouse_rightbtn(AUX_EVENTREC* event);// обработчик
правой клавиши мыши
void CALLBACK keydown(void);//обработчик нажатия клавиши
DOWN void CALLBACK keyup(void);//обработчик нажатия клавиши UP
void CALLBACK keyspace(void);// обработчик нажатия клавиши
 SPACE int main(int argc, char* argv[])
{
        //цветовой режим, удаление невидимых линий и двойная буферизация
        auxInitDisplayMode( AUX_RGBA | AUX_DEPTH | AUX_DOUBLE);
        auxInitPosition(50,10,400,400); //позиция и размеры окна
        auxInitWindow("Вращающийся чайник");//заголовок окна
        auxReshapeFunc(resize);//реакция на изменение размеров окна
        glEnable(GL_ALPHA_TEST);//учет прозрачности
    glEnable(GL_DEPTH_TEST);//удаление невидимых линий
```

```
        glEnable(GL_COLOR_MATERIAL);//синхронное задание цвета
        рисования и материала
        glEnable(GL_BLEND);//разрешение смешивания цветов
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glEnable(GL_LIGHTING);//учет освещения
        glEnable(GL_LIGHT0);//включение нулевого источника света
  //задание положения и направления нулевого источника света
       float pos[4]={0,5,5,1};
        float dir[3]={0,-1,-1};
        glLightfv(GL_LIGHT0, GL_POSITION,
        pos); glLightfv(GL_LIGHT0,
        GL_SPOT_DIRECTION, dir);
        auxIdleFunc(display);//задание анимации
        auxMouseFunc( AUX_LEFTBUTTON,AUX_MOUSEDOWN,
        mouse_leftbtn);
        auxMouseFunc(AUX_RIGHTBUTTON, AUX_MOUSEDOWN,
        mouse_rightbtn);
        auxKeyFunc(AUX_DOWN, keydown);
        auxKeyFunc(AUX_UP, keyup);
        auxKeyFunc(AUX_SPACE, keyspace);
        auxMainLoop(display);//перерисовка окна
        return 0;
}
void CALLBACK resize (int width, int height)
{
        glViewport(0,0,width,height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-5,5,-5,5,2,12);//задание типа проекции
        gluLookAt( 3,0,5, 0,0,0, 0,1,0);//задание позиции
        наблюдателя glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}
void CALLBACK display (void)
{
        //очистка буфера кадра
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        static float angle=0;//угол поворота чайника
        switch (clr_number)
```

```
        {
        case 0: glColor3f(0.5,0.5,0); break;
        case 1: glColor3f(1,0,0); break;
        case 2: glColor3f(0,1,0); break;
        case 3: glColor3f(0,0,1); break;
        default : glColor3f(1,1,1);
        }
        glPushMatrix();
        switch (ax)
        {
        case 0: glRotatef(angle,1,0,0); break;
        case 1: glRotatef(angle,0,1,0); break;
        case 2: glRotatef(angle,0,0,1); break;
        default : glRotatef(angle,0.5,0.5,0);
        }
        auxWireTeapot(2);
        glPopMatrix();
        if (flag)
        {
         angle+=d;
         if (angle>360) angle-=360;
        }
        //копирование содержимого буфера кадра на экран
        glFlush();
        auxSwapBuffers();
}
void CALLBACK mouse_leftbtn( AUX_EVENTREC* event)
{
    if ( ++clr_number==4)
            clr_number=0;
}

void CALLBACK mouse_rightbtn(AUX_EVENTREC* event)
{
        if (flag) flag=false;
                else flag=true;
}
void CALLBACK keydown(void)
```

```
{
        if (d>1) d--;
}
void CALLBACK keyup(void)
{
        if (d<5) d++;

}
void CALLBACK keyspace(void)
{
        if ( ++ax==3) ax=0;
}
```

ПРИЛОЖЕНИЕ 11

Модель солнечной системы

Результат выполнения программы

Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glaux.h>
void CALLBACK resize(int width,int height);
void CALLBACK display(void);
void main()
{
   auxInitPosition( 50, 10, 800, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Модель солнечной системы" );
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
        glEnable(GL_ALPHA_TEST);
   glEnable(GL_COLOR_MATERIAL);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
        float pos[4] = {0,0,0,5};
        float dir[3] = {0,-1,-1};
        float amb0[4]={1,1,0,5};
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   glLightfv(GL_LIGHT0,
   GL_SPOT_DIRECTION, dir);
        glLightfv(GL_LIGHT0, GL_DIFFUSE,amb0);
        float amb[4]={0,0,0,1};
        float def[4]={1,1,1,1};
        float spe[4]={1,1,1,1};
```

```
        float pos1[4] = {10,10,10,5};

        float dir1[3] = {0,1,0};
        glLightfv(GL_LIGHT1, GL_AMBIENT,amb);
        glLightfv(GL_LIGHT1, GL_DIFFUSE, def);
   glLightfv(GL_LIGHT1, GL_SPECULAR, spe);
   glLightfv(GL_LIGHT1, GL_POSITION, pos1);
   glLightfv(GL_LIGHT1,
   GL_SPOT_DIRECTION, dir1);
        glEnable(GL_LIGHT1);
        float pos2[4] = {10,-10,0,7};
        float dir2[3] = {0,1,0};
        glLightfv(GL_LIGHT2, GL_AMBIENT,amb);
        glLightfv(GL_LIGHT2, GL_DIFFUSE, def);
   glLightfv(GL_LIGHT2, GL_SPECULAR, spe);
   glLightfv(GL_LIGHT2, GL_POSITION, pos2);
   glLightfv(GL_LIGHT2,
   GL_SPOT_DIRECTION, dir2);
        glEnable(GL_LIGHT2);
auxIdleFunc(display);
auxMainLoop(display);
}
void CALLBACK resize(int width,int height)
{
  glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-20,20, -10,10,-50,50);
  gluLookAt( 3,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
}
void CALLBACK display(void)
{
        glClear( GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT); static float angle=-360;
   static float ang=-1;
        if (angle<0) glColor3f(1,0,0);else if (angle>0)
        glColor3f(0.5,0.5,1); glRotated(1,0,1,0);
```

```
glPushMatrix();
        glColor3d(2,1,0);
        glTranslated(0,1.2,0);
        auxSolidSphere(2.2);
        glRotated(2,0,1,0);
        glPushMatrix();
        glColor3d(0.5,0.5,0.5);
        glTranslated(16,0,0);
        auxSolidSphere(1);
        glRotated(80,1,0,0);
        glTranslated(0,0,0);
        glColor3d(1,1,1);
        auxSolidTorus(0.3,1.5);
        glPopMatrix();
        glPushMatrix();
        glRotated(ang+angle,0,1,0);
        glColor3d(1.5,0,0);
        glTranslated(-3,0,0);
        auxSolidSphere(0.5);
        glPopMatrix();
        glPushMatrix();
        glRotated(angle,0,1,0);
        glColor3d(1,0,1);
        glTranslated(0,0,-8);
        auxSolidSphere(1);
        glPopMatrix();
        glPushMatrix();
        glRotated(ang,0,1,0);
        glColor3d(0,1,0.5);
        glTranslated(0,0,-10);
        auxSolidSphere(1);
        glRotated(angle*5,0,1,0);
        glColor3d(0.5,0.5,1);
        glTranslated(1.5,0,0);
        auxSolidSphere(0.3);
        glPopMatrix();
        glPushMatrix();
        glColor3d(1,1,1);
        glRotated((ang+angle)*10,0,1,0);
```

```
        glPushMatrix();
        glTranslated(5,0,0);
        auxSolidSphere(0.1);
        glTranslated(-10,0,0);
        auxSolidSphere(0.05);
        glPopMatrix();
        glPushMatrix();
        glTranslated(0,0,5);
        auxSolidSphere(0.1);
        glTranslated(0,0,-10);
        auxSolidSphere(0.05);
        glPopMatrix();
        glPopMatrix();
glPopMatrix();
angle+=5;
if (angle>3600)angle=0;
ang+=1;
if (ang>720)ang=0;
glFlush();
auxSwapBuffers();
}
```

ПРИЛОЖЕНИЕ 12

Модель самолета

Результат выполнения программы

Вращающийся самолет с двигающимися лопастями пропеллера.

Текст программы

1. Листинг выполненной программы:

```
#include <windows.h>
#include <GL/gl.h>
 #include <GL/glu.h>
#include <GL/glaux.h>
#include <math.h>
int i;
float color1[4] = {1,0,0,1};
float color2[4] = {0,1,0,1};
float color3[4] = {0,0,1,1};
AUX_RGBImageRec* photo_image;
unsigned int photo_tex;
void CALLBACK resize(int width,int height)
{
 glViewport(0,0,width,height);
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho(-13,13, -13,12, -5,13);
  gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
  glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{  static int angle=0;
   const speed=5000;
  glClearColor(0.5,0.5,0.5,1);
  glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     photo_image = auxDIBImageLoad("4.bmp");
 glGenTextures(1, &photo_tex);
 glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
 glBindTexture(GL_TEXTURE_2D,photo_tex);
 gluBuild2DMipmaps(GL_TEXTURE_2D, 3, photo_image->sizeX,
 photo_image->sizeY,GL_RGB, GL_UNSIGNED_BYTE,photo_image->data);
 glPushMatrix();
// glEnable(GL_TEXTURE_2D);
```

```
// glEnable(GL_TEXTURE_GEN_S);
// glEnable(GL_TEXTURE_GEN_T);
 //glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP    );
 //glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,GL_SPHERE_MAP);
  glPushMatrix();
  glRotated(90,0,0,1);
  glTranslated(0,5,0);
  glColor3d(1,1,1);
  auxSolidCylinder(1,10);
  glPopMatrix();          //корпус
  glPushMatrix();
  glTranslated(4.2,0,0);
  glColor3d(1,1,1);
  auxSolidSphere(1);
  glColor3d(1,1,1);
  glRotated(90,0,0,1);
  glTranslated(-0.5,-0.2,0);
  auxSolidCylinder(0.5,2);
  glColor3d(1,1,1);
  glRotated(90,1,0,0);
  glTranslated(0,0,1);
 auxSolidCone(0.5,0.5);    //нос
 glColor3d(1,1,1);
 glTranslated(0.5,0,-11.5);
 auxSolidSphere(1);          //задняя часть корпуса
 glColor3d(1,1,1);
 glRotated(90,0,0,1);
 glTranslated(0,-1,0);
 auxSolidCone(0.5,5);
 glColor3d(1,1,1);
 glRotated(90,1,0,0);
 glTranslated(0,-0.08,-1);
 auxSolidCone(1,3);
 glColor3d(1,1,1);
 glRotated(90,0,1,0);
 glTranslated(-1,0,0);
 auxSolidBox(0.3,0.5,4);
 glPopMatrix();              //хвост
 glPushMatrix();
```

```
glRotated(90,0,0,1);
auxSolidBox(0.4,2,12);  //крылья
glTranslated(-0.5,0,3.5);
auxSolidCylinder(0.5,2.2);
glTranslated(0,0,-7);
auxSolidCylinder(0.5,2.2);
glTranslated(0,-1.1,0);
auxSolidSphere(0.5);
glTranslated(0,2.2,0);
auxSolidSphere(0.5);
glTranslated(0,-2.2,7);
auxSolidSphere(0.5);
glTranslated(0,2.2,0);
auxSolidSphere(0.5);
glPopMatrix();
glPushMatrix();
glRotated(90,1,0,0);
glTranslated(0,0.8,1);
auxSolidBox(0.2,0.2,1);
glTranslated(0,-1.6,0);
auxSolidBox(0.2,0.2,1);
glRotated(90,1,0,0);
glColor3d(0,0,0);
glTranslated(0,0.55,0);
auxSolidTorus(0.15,0.25);
glTranslated(0,0,-1.6);
auxSolidTorus(0.15,0.25); //шасси
glRotated(90,0,0,1);
glTranslated(-1,-1.2,-2.8);
    glPushMatrix();
            glRotated(-angle*speed,0,1,0);
                glTranslated(0,0,0.8);
                glRotated(30,0,0,1);
                 glColor3d(0,0,0);
                 auxSolidBox(0.1,0.5,1);
   glPopMatrix();
         glPushMatrix();
                 glRotated(-angle*speed+120,0,1,0);
                 glTranslated(0,0,0.8);
```

```
                glRotated(30,0,0,1);
                glColor3d(0,0,0);
                auxSolidBox(0.1,0.5,1);
        glPopMatrix();
        glPushMatrix();
                glRotated(-angle*speed-120,0,1,0);
                glTranslated(0,0,0.8);
                glRotated(30,0,0,1);
                glColor3d(0,0,0);
                auxSolidBox(0.1,0.5,1);
        glPopMatrix();
  glTranslated(0,0,7);
    glPushMatrix();
          glRotated(-angle*speed,0,1,0);
                glTranslated(0,0,0.8);
                glColor3d(0,0,0);
                glRotated(30,0,0,1);
                auxSolidBox(0.1,0.5,1);
   glPopMatrix();
        glPushMatrix();
                glRotated(-angle*speed+120,0,1,0);
                glTranslated(0,0,0.8);
                glRotated(30,0,0,1);
                glColor3d(0,0,0);
                auxSolidBox(0.1,0.5,1);
        glPopMatrix();
        glPushMatrix();
                glRotated(-angle*speed-120,0,1,0);
                glTranslated(0,0,0.8);
                glRotated(30,0,0,1);
                glColor3d(0,0,0);
                auxSolidBox(0.1,0.5,1);
        glPopMatrix();      //propelleri
BglDisable(GL_TEXTURE_GEN_T);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_TEXTURE_GE
N_S); glPopMatrix();
glRotated(2,0,90,0); //vraschenie
  angle++;
```

```
  auxSwapBuffers();
}
void main()
{

float pos[4] =
{5,5,5,1}; float
sp[4] = {1,1,1,1};
float mat_specular[] = {1,1,1,1};
auxInitPosition( 50, 10, 400, 400);
   auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
   auxInitWindow( "Samolet" );
   auxIdleFunc(display);
   auxReshapeFunc(resize);
   glEnable(GL_DEPTH_TEST);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT3);
         glEnable(GL_LIGHT5);
         glEnable(GL_LIGHT0);
   glLightfv(GL_LIGHT3, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT5, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT0, GL_SPECULAR, sp);
   glLightfv(GL_LIGHT3, GL_DIFFUSE, color1);
   glLightfv(GL_LIGHT5, GL_DIFFUSE, color2);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, color3);
         glLightfv(GL_LIGHT3, GL_POSITION, pos);
         pos[0] = -5; pos[1]=5;
   glLightfv(GL_LIGHT5, GL_POSITION, pos);
         pos[0]=5;pos[1]=-5;
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
         glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
   glMaterialf(GL_FRONT, GL_SHININESS, 128.0);
auxMainLoop(display);
}
```

Приложение А – Образец титульного листа лабораторной работы

Министерство науки и высшего образования Российской Федерации

Лысьвенский филиал федерального государственного бюджетного образовательного учреждения высшего образования

«Пермский национальный исследовательский политехнический университет»

Лабораторная работа

По дисциплине «Компьютерная графика»

На тему: « »

Выполнил:

студент группы

_____

И.О.Фамилия

_____

(Подпись)

Руководитель:

_____

(Подпись)

Лысьва 201__ г.