

# **Лабораторна робота № 1.**

## **Ініціалізація OpenGL та GLUT. Графічні примітиви OpenGL.**

**Мета роботи:** Набути практичних навичок ініціалізації та налаштування параметрів графічного контексту відображення бібліотеки OpenGL. Ознайомитись з основними графічними примітивами бібліотеки побудови тривимірних зображень OpenGL.

**Завдання до роботи:** Розробити програму в якій буде створено контекст графічного пристрою OpenGL та сформовано тривимірне зображення за допомогою основних графічних примітивів, що реалізовані у бібліотеці OpenGL.

### **Теоретичні відомості.**

OpenGL є одним з основних та популярних графічних стандартів ([www.opengl.org](http://www.opengl.org)) для побудови тривимірних зображень, що розроблений компанією Silicon Graphics Inc. ([www.sgi.com](http://www.sgi.com)). Стандарт OpenGL досить поширений серед розробників програмного забезпечення тому, що він підтримується більшістю сучасних операційних систем, має велику функціональність по створенню реалістичного тривимірного зображення та підтримується виробниками графічного апаратного забезпечення.

### **Ініціалізація OpenGL.**

В OpenGL не існує вбудованих функцій для ініціалізації OpenGL. Це пов'язано з тим, що OpenGL є незалежним від платформи графічним інтерфейсом для створення програмного забезпечення (Application Programming Interface – API), а власне ініціалізацію забезпечує операційне середовище.

Однак для того, щоб налаштувати OpenGL під конкретну операційну систему потрібно або використовувати функції по налаштуванню OpenGL для даної операційної системи, або використати бібліотеку яка виконає такі налаштування. Однією з таких крос платформних бібліотек є бібліотека GLUT.

Мінімальна програма з використанням GLUT, яка створює вікно та малює в ньому зображення, складається з наступних частин:

1. Ініціалізація GLUT
2. Встановлення параметрів вікна та створення вікна.

3. Встановлення функцій для малювання та зміни форми вікна.
4. Вхід до головного циклу обробки подій GLUT.

Для ініціалізації GLUT використовується функція:

```
void glutInit(int * argc, char ** argv);
```

Перший параметр є покажчиком на кількість аргументів в командному рядку, а другий - покажчик на масив аргументів. Зазвичай ці значення беруться з головної функції програми:

```
int main(int argc, char **argv)
```

Встановлення параметрів складається з декількох етапів. Перший полягає у визначенні розмірів вікна:

```
void glutInitWindowSize(int width, int height);
```

Перший параметр `width` - ширина вікна в пікселях, другий `height` - висота вікна в пікселях (якщо цю команду опустити, то GLUT сам встановить розміри вікна за замовчуванням, зазвичай 300 на 300).

Далі визначають положення створюваного вікна щодо верхнього лівого кута екрану. Робиться це командою:

```
void glutInitWindowPosition(int x, int y);
```

Також, необхідно встановити режим відображення інформації у вікні, тобто встановити параметри моделі кольорів, буферів екрану, глибини тощо. Для чого в GLUT існує команда:

```
void glutInitDisplayMode(unsigned int mode);
```

Параметр команди `mode`, визначається за допомогою однієї з наступних констант або комбінації цих констант за допомогою побітового АБО.

Константа	Значення
GLUT_RGB	Для відображення графічної інформації використовуються 3 компоненти кольору RGB.
GLUT_RGBA	Те ж що і RGB, але додатково використовується четверта компоненти ALPHA (прозорість).
GLUT_INDEX	Колір задається не за допомогою RGB компонентів, а за допомогою палітри. Використовується для старих дисплеїв, де кількість кольорів невелика, наприклад 256.
GLUT_SINGLE	Виведення у вікно здійснюється з використанням 1 буфера. Зазвичай використовується для виведення статичної інформації.

GLUT_DOUBLE	Виведення у вікно здійснюється з використанням 2 буферів. Застосовується для створення анімованих зображень, щоб виключити ефект мерехтіння.
GLUT_ACCUM	Використовувати буфер накопичення (Accumulation Buffer). Цей буфер застосовується для створення спеціальних ефектів, наприклад відображення та тіней.
GLUT_ALPHA	Використовувати ALPHA буфер. Цей буфер, як уже говорилося використовується для завдання 4-го компоненту кольору - ALPHA. Звичайно застосовується для таких ефектів як прозорість об'єктів і анти-аліасінгу.
GLUT_DEPTH	Створити буфер глибини. Цей буфер використовується для відсікання невидимих частин зображення в 3D просторі при виведенні на плоский екран монітора.
GLUT_STENCIL	Буфер трафарету використовується для таких ефектів як вирізання частини фігури. Наприклад, наклавши прямокутний трафарет на стіну будинку, ви отримаєте вікно, через яке можна побачити що знаходиться всередині будинку.
GLUT_STEREO	Це значення використовується для створення стерео-зображень. Використовується рідко, тому що для перегляду такого зображення потрібна спеціальна апаратура.

Приклад використання:

```
void glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
```

## Створення вікна.

Після того як встановлено параметри вікна необхідно його створити. Для створення вікна використовується команда:

```
int glutCreateWindow(const char *title);
```

Ця команда створює вікно з заголовком title та повертає HANDLER вікна у вигляді цілого числа. HANDLER вікна зазвичай використовується для подальших операцій над цим вікном.

## Крок 1 - Вікно OpenGL

### Створення проекту

Створюємо проект Win32 Console.

Нам необхідно підключити, для компіляції даного проекту 2 бібліотеки – Opengl32.lib та glut32.lib. Дістатися до установки бібліотек можна з допомогу Project Setting - Link - Object library module.

У цих бібліотеках не сам код, а посилання на DLL. OpenGL тепер входить в постачання Windows NT, Windows 98.

У цьому варіанті ми отримаємо окреме вікно OpenGL. На ньому і будемо тренуватися.

### Створюємо код

```
#include <stdafx.h>
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE); //set the display to
Double buffer
    glutInitWindowSize(300, 300);
    glutCreateWindow(argv[0]);
    glutMainLoop(); //call the main loop
    return 0;
}
```

### Опис

OpenGL є ймовірно найпоширенішим індустріальним програмним інтерфейсом API для розробки 3D-додатків. Він являє собою відкритий стандарт, створений фахівцями SGI і перебуває у віданні спеціального комітету Architecture Review Board, в роботі якого беруть участь і SGI і Microsoft.

Якщо ви запустите і виконайте програму, то з'явиться і зникне вікно, швидкість в залежності від швидкості вашого ПК.

### **Встановлення функцій малювання та зміни форми вікна.**

Після того, як вікно, в яке буде виводиться графічна інформація, підготовлено та створено, необхідно зв'язати з ним процедури, які будуть відповідати за виведення графічної інформації, відслідковувати зміну розмірів вікна, відслідковувати натискання клавіш тощо.

Функція яка, відповідає за малювання (вона завжди буде викликатися операційною системою, щоб перемалювати вміст вікна) задається командою:

```
void glutDisplayFunc(void (*func)(void));
```

Параметр цієї функції - це покажчик на функцію, яка буде відповідати за малювання у вікні. Наприклад, для того щоб функція `void Draw(void)`, визначена в програмі відповідала за малювання у поточному вікні, треба приєднати її до GLUT наступним чином:

```
glutDisplayFunc(Draw);
```

Функцію, яка відслідковує зміну розмірів вікна визначають наступною командою:

```
void glutReshapeFunc(void (*func)(int width, int height));
```

Параметр - це покажчик на функцію, що відповідає за зміну розмірів вікна, яка повинна приймати два параметри `width` і `height`, відповідно нову ширину і висоту вікна.

### **Вхід в головний цикл GLUT.**

Головний цикл GLUT це цикл, який забезпечує взаємозв'язок між операційною системою та тими функціями, які відповідають за вікно, отримують інформацію від пристроїв введення / виводу. Для того, щоб перейти в головний цикл GLUT, треба виконати команду:

```
void glutMainLoop(void);
```

Перш ніж використовувати з GLUT необхідно до програми підключити файл заголовку: `#include <gl/glut.h>;`

## Налаштування параметрів виведення зображення.

OpenGL містить велику кількість функцій, за допомогою яких можна виводити двох та трьох вимірні графічні примітиви, управляти їх властивостями, способами виведення параметрами освітлення, створення текстури та інше.

`glViewport(x, y, width, height)` – встановлює область виведення, у яку OpenGL буде виводити зображення.

`glClearColor(r, g, b, a)` – встановлює колір, яким буде заповнюватися вікно при очищенні. У цієї процедури - 4 параметри, що відповідає RGBA. Замість неї можна використати `glClearIndex(0.0)`, однак ця функція встановлює індекс кольору в палітрі.

`glClearDepth()`, `glClearAccum()`, `glClearStencil()` – встановлює поточне значення для очищення буферу глибини, акумулятора та трафарету відповідно.

`glClear(mask)` – очищує буфери:

`GL_COLOR_BUFFER_BIT` – кольорів;

`GL_DEPTH_BUFFER_BIT` – глибини;

`GL_ACCUM_BUFFER_BIT` – накопичування;

`GL_STENCIL_BUFFER_BIT` – шаблону.

Приклад очистки декількох буферів:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
```

```
glClearDepth(1.0);
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

`glFinish()` – вказує на закінчення формування зображення з примітивів.

## Примітиви

Будь-яку тривимірну фігуру, яка б складна вона не була, можна розбити на прості складові. Ці складові називаються примітивами. Примітиви визначаються однієї або послідовністю декількох точок, що в OpenGL задаються всередині командних дужок `glBegin/glEnd`.

```
void glBegin(GLenum mode);
```

```
void glEnd();
```

Послідовність точок, що вводяться в між командами `glBegin / glEnd` формують примітив, тип якого заздалегідь визначається як параметр `mode` функції `glBegin`:

GL_POINTS	Кожна вершина – окрема точка.
GL_LINES	Кожна пара вершин – окрема лінія. Якщо число вершин непарне, то остання вершина ігнорується.
GL_LINE_STRIP	Послідовність зв'язаних відрізків. Перші дві вершини - перший відрізок. Третя вершина визначає другий відрізок з початком вкінці першого та кінцем у цій вершині і т.д.
GL_LINE_LOOP	Аналогічно режиму GL_LINE_STRIP за виключенням того, що остання вершина з'єднується відрізком з першою.
GL_TRIANGLES	Кожна трійка вершин – окремий трикутник (рис.1-с).
GL_TRIANGLE_STRIP	Група зв'язаних трикутників. Перші три вершини – перший трикутник. Друга, третя і четверта вершини – другий трикутник і т.д. (рис.1-а).
GL_TRIANGLE_FAN	Група зв'язаних трикутників. Перші три вершини – перший трикутник. Перша, друга і четверта вершини - другий трикутник і т.д. (рис.1-б).
GL_QUADS	Кожні чотири вершини – окремий чотирикутник (рис.2-б).
GL_QUAD_STRIP	Група зв'язаних чотирикутників. Перші чотири вершини - перший чотирикутник. Третя, четверта, п'ята і шоста вершини - другий чотирикутник і т.д. (рис.2-а).
GL_POLYGON	Малює окремий опуклий багатокутник (один).

Для кожного примітива визначено мінімальне число вершин. У випадку, якщо зазначене число вершин менше ніж мінімальне для даного примітива, то примітив не відображається.

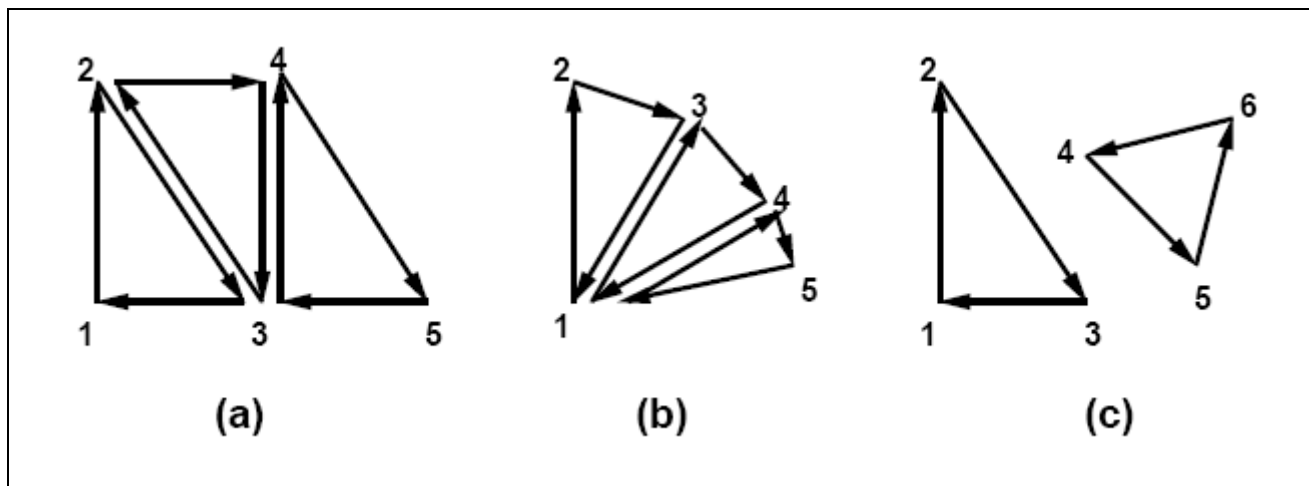


Рис. 1. Режими побудови трикутників:  
a – GL\_TRIANGLE\_STRIP; b – GL\_TRIANGLE\_FAN; c – GL\_TRIANGLES.

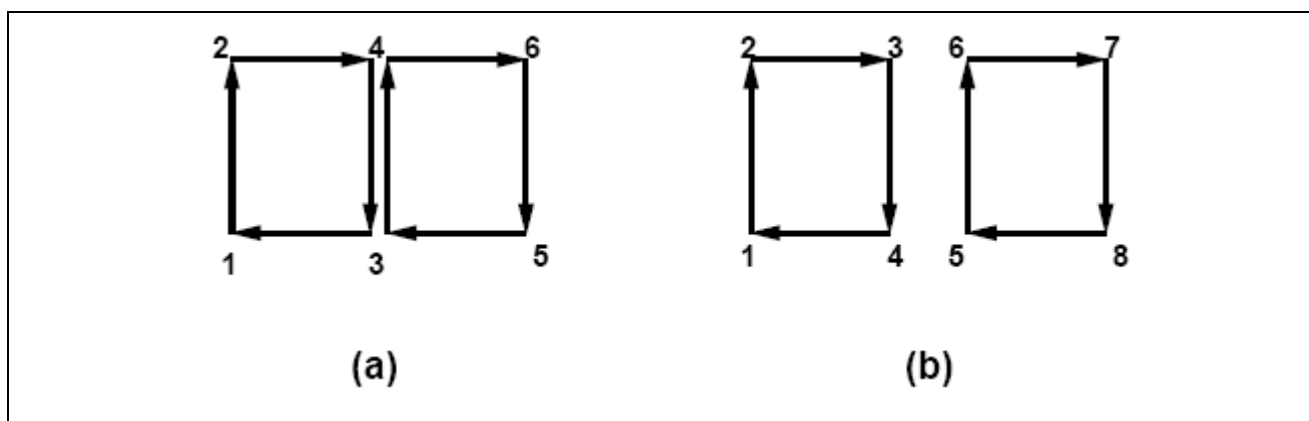


Рис. 2. Режими побудови прямокутників:  
a – GL\_QUAD\_STRIP; b – GL\_QUADS.

Вершини примітивів визначаються за допомогою процедури `glVertex`. `glVertex[2 3 4][s i f d][v](coord)`. Де позначення `[2 3 4]` вказує на доповнення до назви процедури `glVertex`: кожна з цифр визначає ім'я процедури з відповідною кількістю параметрів. Приставка `[s i f d]` до імені процедури вказує на тип даних, які можуть бути передані у відповідну процедуру, а приставка `[v]` вказує на те що аргумент є вектором координат. Так, наприклад при виклику процедури `glVertex3d` потрібно передати три координати дійсного типу подвійної точності.

В загальному випадку вершина визначається чотирма координатами -  $x$ ,  $y$ ,  $z$  і  $w$ . Значення за замовчуванням  $z = 0$  та  $w = 1$ , тобто коли програміст викликає, наприклад, `glVertex2f(1.0, 1.0)` то насправді викликається `glVertex4f(1.0, 1.0, 0.0, 1.0)`.



## Крок 2 - Лінія 2D

Створюємо код

```
#include "stdafx.h"
#include "windows.h"
#include "GL/gl.h"
#include "GL/glu.h"
#include "GL/glut.h"

void display (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin (GL_LINES);
    glVertex2f (0,0);
    glVertex2f (100,100);
    glEnd ();
    glFlush ();
}

int main (int argc, char ** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE); //set the display to
Double buffer
    glutInitWindowSize(300, 300);
    glutCreateWindow(argv[0]);
    glutDisplayFunc(display); //use the display function to
draw everything
    glutMainLoop();
    return 0;
}
```

## Опис

Будь-яка послідовність команд з малювання, це прохід по вершинах. Вершини визначаються від `glBegin` (тип) до `glEnd` (). Між цими командами встановлюються вершини функцією `glVertex2f` (x, y) для 2D точок.

glFlush () промальовує екран. Без цієї команди ви нічого не побачите. Спробуйте.

### Крок 3

Вершиною є точка в тривимірному просторі, координати якої можна задавати наступним чином:

```
void glVertex [2 3 4] [s i f d] (type coords)
void glVertex [2 3 4] [s i f d] v (type * coords)
```

Координати точки задаються максимум чотирма значеннями: x, y, z, w, при цьому можна вказувати два (x, y) або три (x, y, z) значення, а для решти змінних в цих випадках використовуються значення за замовчуванням: z = 0, w = 1. Як вже було сказано вище, число в назві команди відповідає числу явно задаються значень, а подальший символ - їх типу.

Координатні осі розташовані так, що точка (0,0) знаходиться в лівому нижньому кутку екрана, вісь x направлена вліво, вісь y-вгору, а вісь z-з екрану. Це розташування осей світової системи координат, в якій задаються координати вершин об'єкта, інші системи координат будуть розглянуті нижче.

Однак, щоб задати яку-небудь фігуру одних координат вершин недостатньо, і ці вершини треба об'єднати в одне ціле, визначивши необхідні властивості. Для цього в OpenGL використовується поняття примітивів, до яких відносяться точки, лінії, пов'язані чи замкнуті лінії, трикутники і так далі. Завдання примітиву відбувається всередині командних дужок:

```
void glBegin (GLenum mode)
void glEnd (void)
```

### Атрибути примітивів

З кожною вершиною будь-якого примітиву зв'язані наступні атрибути:

Атрибут	Команда
Поточний колір – колір вершини (остаточний колір	glColor[3 4][b s i f d ub us ui][v](args).

вираховується з урахуванням освітлення)	
Поточні координати текстури – координати текстури, що відповідають цій вершині	GLTexCoord
Оточна нормаль – вектор нормалі, що відповідає даній вершині	GLNormal

#### Атрибути точок:

Атрибут	Команда
Розмір точки	glPointSize (GLfloat size)
Параметрами згладжування антиаліасінг	glEnable (GL_POINT_SMOOTH) glDisable (GL_POINT_SMOOTH)

#### Атрибути ліній:

Атрибут	Команда
Ширина лінії	glLineWidth(GLfloat width)
Параметрами згладжування антиаліасінг	glEnable (GL_LINE_SMOOTH) glDisable (GL_LINE_SMOOTH)
Штрихування лінії	glLineStipple(GLint factor, GLushort pattern) glEnable (GL_LINE_STIPPLE) glDisable (GL_LINE_STIPPLE)

#### Атрибути трикутників, чотирикутників. багатокутників:

Атрибут	Команда
Параметрами згладжування антиаліасінг	glEnable (GL_POLYGON_SMOOTH) glDisable (GL_POLYGON_SMOOTH)
Штрихування лінії	glPolygonStipple (GLubyte *pattern) glEnable (GL_POLYGON_STIPPLE) glDisable (GL_POLYGON_STIPPLE)
Режим малювання	glPolygonMode (GLenum face, GLenum mode)

Правило розрізнення граней	glFrontFace (GLenum mode) glEnable (GL_CULL_FACE) glDisable (GL_CULL_FACE)
----------------------------	--

## Крок 4 - розфарбовуємо трикутник

Різним вершинам можна призначати різні кольори і тоді буде проводитися лінійна інтерполяція кольорів по поверхні примітиву. Для управління режимом інтерполяції кольорів використовується команда void glShadeModel (GLenum mode) виклик якої з параметром GL\_SMOOTH включає інтерполяцію (значення за замовчуванням), а з GL\_FLAT відключає. Наприклад, щоб намалювати трикутник з різними кольорами в вершинах, досить написати:

```
glBegin (GL_TRIANGLES);
glColor3f (1.0, 0.0, 0.0); //червоний
glVertex3f (0.0, 0.0, 0.0);
glColor3f(0,1,0); //зелений
glVertex3f (1.0, 0.0, 0.0);
glColor3f(0.0,0.0,1.0); //блакитний
glVertex3f (1.0, 1.0, 0.0);
glEnd ();
glFlush ();
```

## Списки примітивів

Список - це деякий набір команд OpenGL, який можна створити один раз, наприклад, перед початком роботи програми, а потім їм користуватися викликаючи його. За логікою він аналогічний процедурою (або функції, як хочете). Відмінність його від процедури в тому, що при формуванні списку бібліотека зберігає його в якомусь своєму, оптимізованому форматі і виклики списку призводять до значно меншим обчислювальним витратам ніж виклик функції, що робить теж саме. Особливо добре це помітно на картах, де операції зі списками реалізовані апаратно.

Давайте подивимося як створити список:

```
// Відкриваємо список зображень (без виконання) під номером 2
glNewList(1, GL_COMPILE);
glColor3f(1.0f, 0.0f, 0.0f); // Задаємо колір створюваного
примітива
glBegin(GL_POLYGON); // Задаємо сам примітив
    glVertex3f( 1.0f, 1.0f, 0.0f);
    glVertex3f(-1.0f, 1.0f, 0.0f);
    glVertex3f(-1.0f, -1.0f, 0.0f);
    glVertex3f( 1.0f, -1.0f, 0.0f);
glEnd();
glEndList(); // Закриваємо список зображень під номером 1
glCallList(1); // викликаємо список
```

Для того, щоб занести в список деяку послідовність команд треба помістити їх між командними дужками `glNewList` і `glEndList`. Перший аргумент команди `glNewList` - номер списку. Ви можете використовувати велику кількість списків у своїй програмі і кожен з них буде мати свій номер. У принципі можна самому задавати числа, але для того щоб уникнути накладень рекомендується використовувати спеціальну команду, яка генерує номери списків і перший номер повертає у вашу змінну. Як аргумент вона приймає кількість номерів списків, яке ви хочете отримати

У OpenGL реалізовано технологію зберігання послідовності формування примітивів, для цього використовуються списки. Для створення списку потрібно викликати функцію:

```
void glNewList(GLuint list, GLenum mode);
```

Параметр `list` - беззнакове ціле число, що використовується для ідентифікації списку. Параметр `mode` - це режим формування зображення, що може приймати наступні значення:

`GL_COMPILE` – команди просто запам'ятовуються без виконання;

`GL_COMPILE_AND_EXECUTE` – команди спочатку виконуються, після чого заносяться до списку.

Всі команди, розташовані після виклику процедури `glNewList` заносяться в список доти, поки не викликається команда:

```
void glEndList(void);
```

Для виклику списку використовується команда:

```
void glCallList(GLuint list);
```

де list - беззнакове ціле число, що визначає список.

Якщо у вас існує кілька списків, то їх можна викликати одночасно. Для цього використовується команда:

```
void glCallLists(GLsizei n, GLenum type, const GLvoid  
*lists);
```

де n - кількість списків, lists - покажчик на масив списків, а type - тип значень у списку.

Перед викликом glCallLists можна викликати процедуру glListBase(base), що додає постійне ціле число base, яке визначає зсув до кожного імені списку з масиву lists. Дозволено викликати процедури glCallList і glCallLists з інших списків, але рівень вкладеності не перевищує 64 (конкретне значення залежить від поточної реалізації OpenGL).

### **Порядок виконання роботи.**

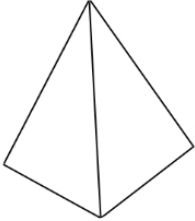
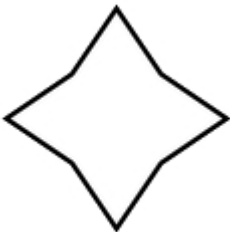
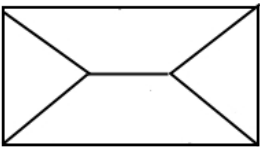
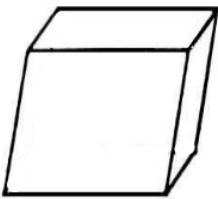

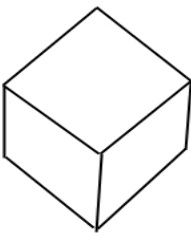


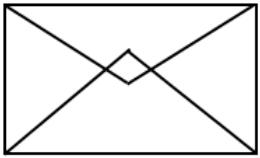
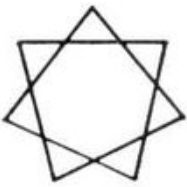

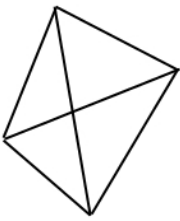

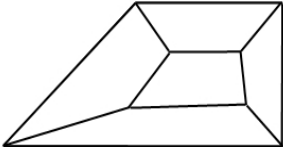
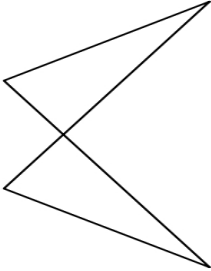
- 1) Створити консольну програму, що містить код для ініціалізації бібліотеки GLUT, встановлення параметрів вікна та створення вікна, встановлення функцій для малювання та зміни форми вікна (функції обробки клавіатури за бажанням), вхід до головного циклу обробки подій GLUT.
- 2) Створити всі найпростіші графічні примітиви, визначити атрибути елементів графічних примітивів:
  - a. всі типи примітивів: колір;
  - b. точки: розмір точок, режим відображення зі згладжуванням та без згладжування;
  - c. лінії: ширина лінії, штрихування, режим відображення зі згладжуванням та без згладжування;
  - d. трикутники, чотирикутники, багатокутники: режим відображення зі згладжуванням та без згладжування, режим малювання (glPolygonMode), штриховка.
- 3) Створити зображення за допомогою примітивів згідно варіанту за таблицею №1.
- 4) Створити списки зображень (для кожного типу примітивів власний список) Списки зображень формуються на етапі ініціалізації програми. В функції малювання сформувані зображення за раніше створеними списками.

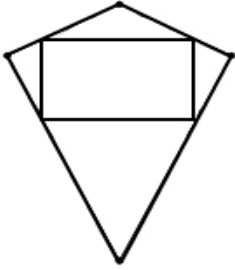
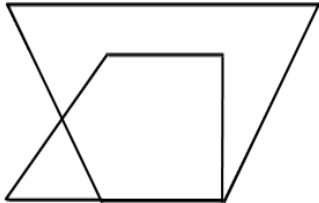
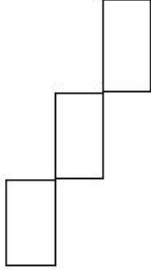
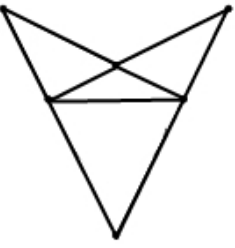
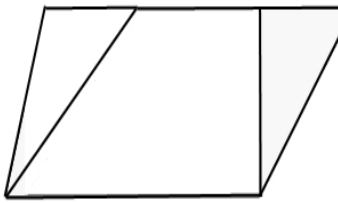
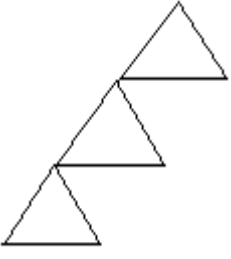

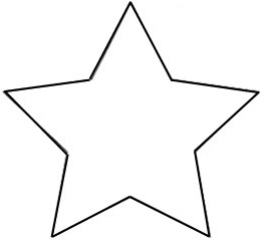
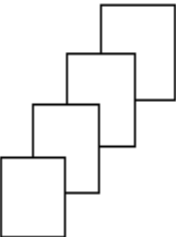
Для отримання оцінки задовільно – завдання 1,2

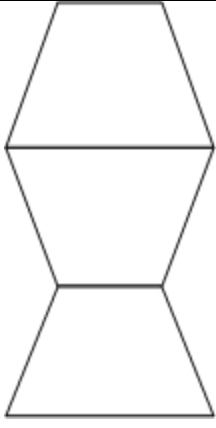
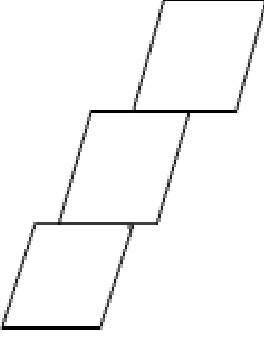
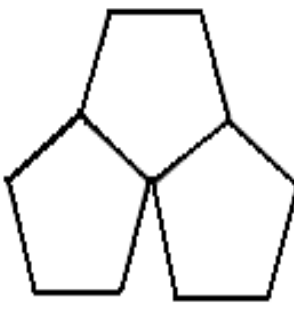
Для отримання оцінки добре – завдання 1,2,3

Для отримання оцінки відмінно – завдання 1,2,3,4

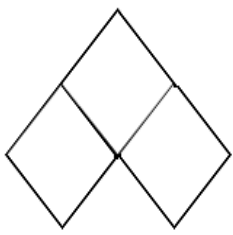
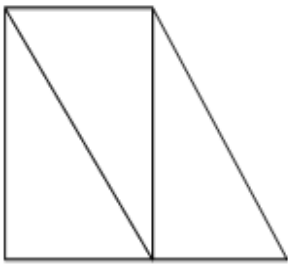
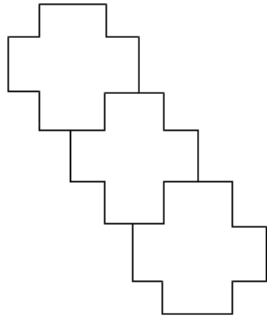
Таблиця №1

1.		9.		17.	
2.		10.		18.	
3.		11.		19.	
4.		12.		20.	
5.		13.		21.	

6.		14.		22.	
7.		15.		23.	
8.		16.		24.	

25.		27.		29.	
-----	---	-----	---	-----	---



26.		28.		30.	
-----	---	-----	---	-----	---

## Додаток № 1

Приклад написання програми до пункту 1.

```
#include <stdafx.h>
#include <windows.h>

#include <GL/gl.h>
#include <GL/glut.h>
#include <math.h>

void init(void)
{
    glShadeModel(GL_SMOOTH); //set the shader to smooth shader
}

void display(void)
{
    glClearColor(0.2, 0.3, 1.0, 1.0); // clear the screen to blue
    glClear(GL_COLOR_BUFFER_BIT);      // clear the color
    glutSwapBuffers(); // swap the buffers
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); // set the matrix
to projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity(); //set the matrix back to model
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void keyboard(unsigned char key, int x, int y)
{

```

```

        if(key==27)
        {
            exit(0); // quit the program
        }
    }

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE); //set the display to Double
buffer
    glutInitWindowSize(300, 300);
    glutCreateWindow(argv[0]);

    init(); // call the init function

    glutDisplayFunc(display); //use the display function to draw
everything
    glutReshapeFunc(reshape); //reshape the window accordingly
    glutKeyboardFunc(keyboard); //check the keyboard
    glutMainLoop(); //call the main loop
    return 0;
}

```