Лабораторна робота № 2

Геометричні перетворення та проекції

Мета роботи**:* Ознайомитися та навчитися працювати з матрицями афінних перетворень і проекцій в ***OpenGL, навчитися будувати та передавати в ***OpenGL*** матриці довільних перетворень. Створити програму, яка ілюструє виконання афінних перетворень над об'єктами тривимірного простору та їх проектування на площину.

Завдання до роботи: Розробити програму в якій будуть використані засоби ***OpenGL*** для геометричних перетворень та проекцій, сформовано тривимірне зображення за допомогою основних графічних примітивів, що реалізовані у бібліотеці ***OpenGL*** та використано функції про обертанню, масштабуванню та переносу забраження .

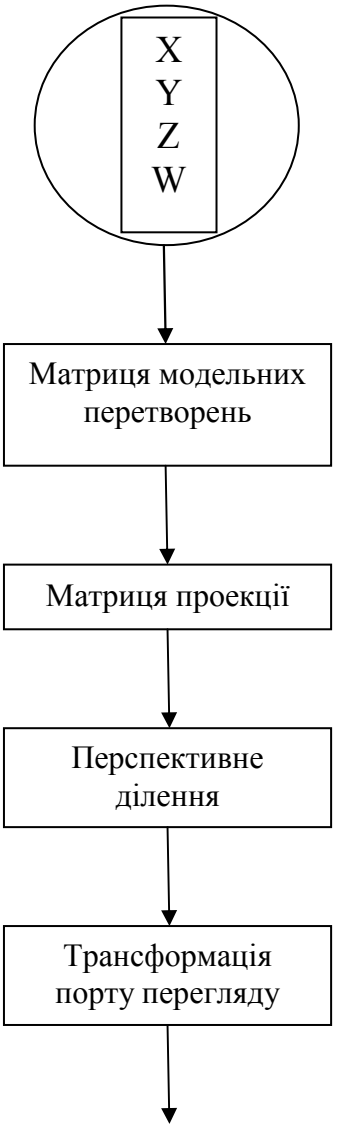
Матричні операції.

В ***OpenGL*** передбачені три типи матриць: виду моделі, проекцій і текстури. Всі вони мають розмірність (4 на 4). Для того, щоб з матрицею можна було працювати, її потрібно зробити поточною, для цього передбачена команда `glMatrixMode(mode: GLenum)`. Для зміни типу проектування, положення або напрямку камери, використовують матрицю проектування ***(GL_PROJECTION)*** для зміни масштабу, кута обертання, зсуву зображення використовують матрицю модельного представлення ***(GL_MODELVIEW).*** Після того, як матриця встановлена, необхідно визначити її елементи. Для досягнення цієї мети можна використовувати команди:

`void glLoadidentity(void)`	Заміна поточної матриці на одиничну
`void glLoadMatrix[fd](GLtype *m)`	Завантаження матриці m в поточну
`void glMultMatrix[fd](GLtype *m)`	Множення поточної матриці на матрицю m
`void glPushMatrix(void);`	Збереження поточної матриці в стек
`void glPopMatrix(void);`	Виштовхування матриці з вершини стеку в поточну матрицю

- Тривимірні координати об'єкту перетворяться у позиції пікселів на екрані серією з трьох операцій. Трансформації (перетворення), представлені перемножуванням матриць, включають модельні, видові і проекційні операції. До таких операцій відносяться обертання, перенесення, масштабування, віддзеркалення, ортографічне і перспективне проектування.
- Зазвичай для відображення сцени використовується комбінація з декількох трансформацій. Оскільки сцена відображується в прямокутному вікні, об'єкти (або їх частини), що знаходяться зовні вікна, мають бути відсічені. У тривимірній комп'ютерній графіці відсікання виробляється шляхом відкидання об'єктів з одного боку відсікаючої плоскості.
- Нарешті, має бути встановлене відповідність між перетвореними координатами і екранними пікселями. Цей процес відомий, як трансформація порту перегляду.

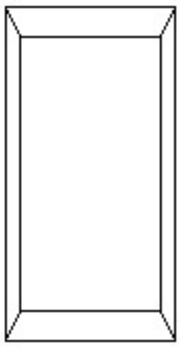
Малюнок 3-2 демонструє порядок, в якому ці операції виробляються вашим комп'ютером. Малюнок 3-2. Етапи перетворення вершин



Щоб задати видове, модельне або проекційне перетворення, заповнюється матриця M розмірністю 4x4 елемента. Потім ця матриця множиться на координати кожної вершини, у результаті чого й відбувається перетворення цих координат v' = Mv (вершина завжди має 4 координати (x, y, z, w), в більшості випадків w = 1, для двовимірних зображень z = 0) . Видові й модельні перетворення поряд з координатами вершин автоматично застосовуються до нормалей до поверхні в цих вершинах. Нормалі використовуються тільки у видових координатах. Це робиться для того, щоб зберегти зв'язок між вектором нормалі й верховими даними.

Приклад 3-1 малює куб, який масштабується модельною трансформацією. Видова трансформація позиціонує і наводить камеру на те місце, де буде намальований куб. Проекційна трансформація і трансформація порту перегляду також вказані. Після прикладу слідують розділи, що розбирають приклад 3-1. У них дається короткий опис команд трансформацій.

Малюнок 3-3. Трансформований куб



```
#include <glut.h>
//Ініціалізація
void init (void)
{
glClearColor(0.0,0.0,0.0,0.0);
glShadeModel(GL_FLAT);
}
//Відображення
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,1.0);
```

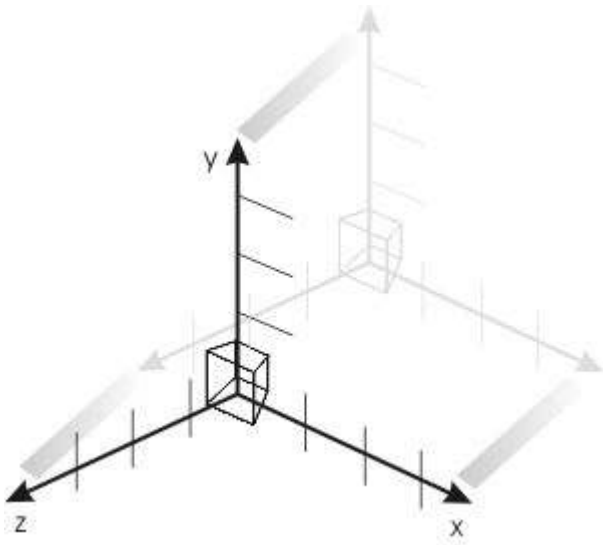
```
//Очистити матрицю
glLoadIdentity();
//Видова трансформація (камера)
gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
//Модельна трансформація
glScalef(1.0,2.0,1.0);
glutWireCube(1.0);
glFlush();
}
// Зміна розмірів вікна
void reshape (int w, int h)
{
glViewport(0,0,(GLsizei) w, (GLsizei) h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
glMatrixMode(GL_MODELVIEW);
}
int main (int argc, char** argv)
{}
```

Команда визначення об’єму видимості ***glFrustum*** обчислює матрицю, що виконує перспективне проектування, та множить на неї поточну матрицю проекції (зазвичай одиничну). Зауважимо, що обсяг видимості використовується для відсікання об'єктів що лежать поза ним (наприклад чотири сторони піраміди, її основа і вершина (точніше, верхня сторона) відповідають шести відтинаючим площинам. Об'єкти або частини об'єктів поза цими площинами відсікаються і не виводяться у фінальному зображенні. ***glFrustum()*** не вимагає визначення симетричного об’єму видимості.)

Функція переносу

```
void glTranslate{fd} (TYPE x, TYPE y, TYPE z);
```

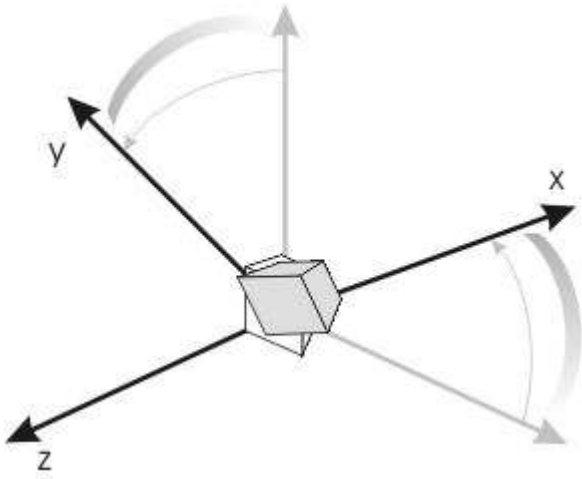
Множить поточну матрицю на матрицю, що пересуває (що переносить) об'єкт на відстані x, y, z, передані як аргументи команди, по відповідних осях (чи переміщає локальну координатну систему на ті ж відстані).
На малюнку 3-5 зображений ефект команди *glTranslate*()*.



Функція повороту

```
void glRotate{fd} (type angle, TYPE x, TYPE y, TYPE z);
```

Множить поточну матрицю на матрицю, яка повертає об'єкт (чи локальну координатну систему) в напрямі проти годинникової стрілки навколо променя з початку координат, що проходить через точку (x, y, z). Параметр angle задає кут повороту в градусах. Результат виконання *glRotatef(45.0, 0.0, 0.0, 1.0)*, тобто поворот на 45 градусів навколо осі z, показаний на малюнку 3-6.



Помітьте, що чим далі об'єкт від осі обертання, тим більше орбіта його повороту і тим помітніше сам поворот. Також звернете увагу на те, що виклик ***glRotate*()*** з параметром ***angle*** рівним 0 не має ніякого ефекту.

Функція масштабування

Матричний вираз тривимірного масштабування точки P = (x, y, z) відносно початку координат є простим розширенням двовимірного масштабування. В матрицю трансформації просто додається параметр, що відповідає за масштабування по осі z:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

або в матричній формі:

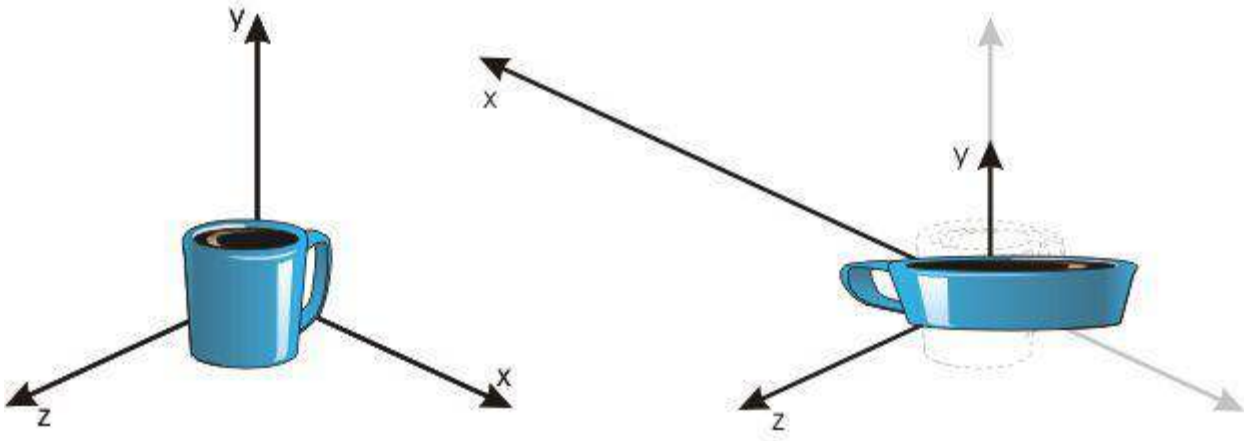
$$P' = MS \cdot P$$

де Sx, Sy, Sz - коефіцієнти масштабування по осях, а MS – матриця масштабування.

```
void glScale{fd} (TYPE x, TYPE y, TYPE z);
```

Множить поточну матрицю на матрицю, яка розтягує, стискує або відбиває об'єкт уподовж координатний осей. Кожна x -, y - і z - координата кожної точки об'єкту буде помножена на відповідний аргумент x, y або z команди ***glScale*().*** При розгляді перетворення з точки зору локальної координатної системи, осі цієї системи розтягуються, стискуються або відбиваються з урахуванням чинників x, y і z, і асоційований з цей системою об'єкт міняється разом з нею.

На малюнку показаний ефект команди ***glScalef(-2.0,0.5,1.0);***



glScale*() - це єдина з трьох команд модельних перетворень, змінююча розмір об'єкту : масштабування з величинами більше 1.0 розтягує об'єкт,

використання величин менше 1.0 стискує його. Масштабування з величиною - 1.0 відбиває об'єкт відносно осі або осей. Одиничними аргументами (тобто аргументами, що не мають ефекту) є (1.0, 1.0, 1.0). Взагалі слід обмежувати використання ***glScale*()*** тими випадками, коли це дійсно необхідно. Використання ***glScale*()*** знижує швидкодію розрахунків освітленості, оскільки вектора нормалей мають бути нормалізовані наново після перетворення.

Зауваження: Величина масштабування рівна 0 призводить до колапсу усіх координат об'єкту по осі або осям до 0. Зазвичай це не є хорошою ідеєю, оскільки така операція не може бути обернена. Говорячи математично, матриця не може бути обернена, а зворотні матриці потрібні для багатьох розрахунків, пов'язаних з освітленням. Іноді колапс координат все ж має сенс: розрахунок тіней на плоскій поверхні - це типовий приклад застосування колапсу. Вобщем, якщо координатна система має бути піддана колапсу, слід використовувати для цього проекційну матрицю, а не видову.

Проекційні трансформації

Призначення проекційного перетворення полягає у визначенні об’єму видимості. Об’єм видимості визначає, як об'єкт проектується на екран (з використанням перспективної або паралельної проекції), він також визначає, які об'єкти або частини об'єктів будуть відсічені в результуючому зображенні.

Перспективна проекція.

Найбільш зрозумілою характеристикою перспективної проекції є зменшення віддалених частин зображення: чим далі об'єкт знаходиться від камери (точки спостереження), тим меншим він буде у фінальному зображенні. Це відбувається тому, що об’єм видимості перспективної проекції має форму усіченої піраміди (піраміди, верхівка якої відрізана площиною, паралельною її основі). Об'єкти, потрапляють в обсяг видимості проектуються з вершини піраміди, де знаходиться точка спостереження. Ближчі до точки спостереження об'єкти виходять більшими, оскільки вони займають пропорційно більший простір обсягу видимості.

Об'єкти, що знаходяться далі виявляються меншими, оскільки вони знаходяться у вужчій частини усіченої піраміди об’єму видимості. Даний метод проектування використовується для анімації, візуальної симуляції і в будь-яких інших програмах, які претендують на деяку реалізм, так як перспективне проектування схоже на те, як бачить людське око (або камера).

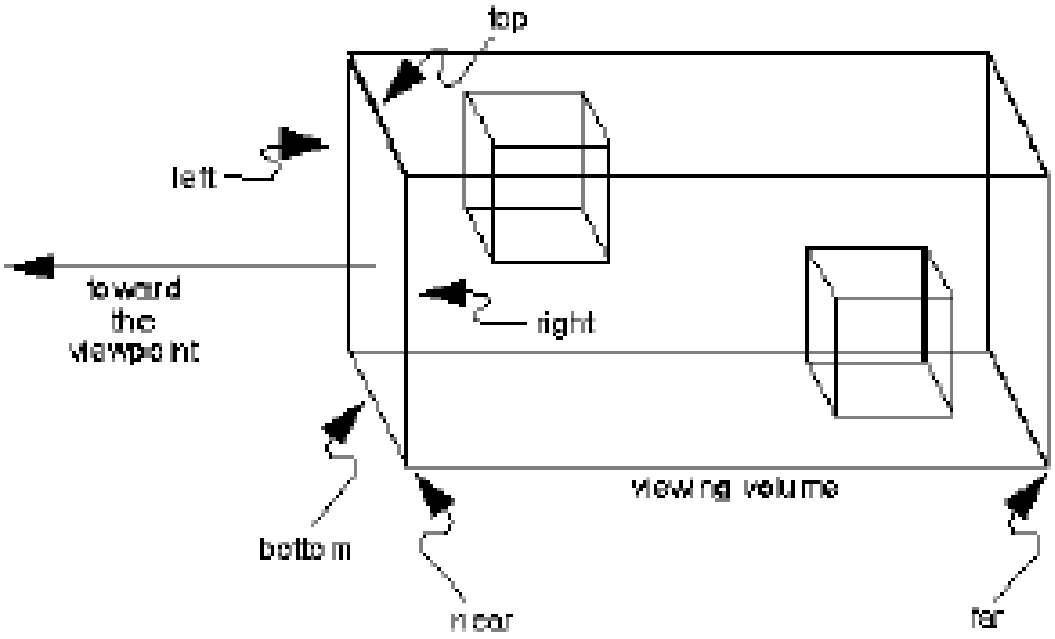
Ортографічна проекція.

При використанні ортографічної проекції областю видимості є прямокутний паралелепіпед. На відміну від перспективної проекції розмір області видимості не змінюється від одного кінця до іншого, таким чином, відстань від камери не впливає на розмір об'єктів в результуючому зображенні. Така проекція використовується у випадку, якщо важливі реальні розміри об'єктів щодо один одного і точність відображення кутів між ними.

Команда ***glOrtho*** створює матрицю для паралельного об’єму видимості ортографічної проекції та множить на неї поточну матрицю:

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

Параметри ***(left, bottom,-near)*** та ***(right, top,-near)*** задають точки на ближній площині відсікання, які будуть спроектовані відповідно на нижній лівий і верхній правий кути порту перегляду. ***(left, bottom,-far)*** і ***(right, top,-far)*** це точки на дальній площині відсікання, які будуть спроектовані на ті ж кути порту перегляду.



Значення для ***far*** і ***near*** можуть бути позитивними, негативними або навіть нульовими, проте вони не повинні бути однаковими. Якщо ніякі інші перетворення не використовуються, напрямок проектування збігається з віссю z, а напрямок огляду – з її негативним напрямком. Для спеціального випадку, коли двовимірне зображення проектується на двовимірний екран,

можна використовувати функцію ***gluOrtho2D()*** з бібліотеки утиліт. Ця функція ідентична команді ***glOrtho(),*** але вона припускає, що всі z - координати об'єктів лежать в діапазоні від 1.0 до -1.0. Якщо зображаються двовимірні об'єкти із застосуванням двовимірних версій верхових команд, всі їхні z – координати дорівнюють нулю, таким чином, жоден з об'єктів не відсікається через свої z - координат.

```
void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom,
GLdouble top).
```

Створює матрицю для проектування двовимірного зображення на екран та множить на неї поточну матрицю. Область відсікання є прямокутником з нижнім лівим кутом в ***(left, bottom)*** і правим верхнім кутом в ***(right, top).***

Примітка. Невідповідність між просторовими об'єктами та плоскими зображеннями усувається шляхом введення проекцій, що відображають об'єкти на картинній площині. Перед завданням матриці проекцій необхідно встановити параметри області виводу, для цього призначена команда:

```
glViewport(x, y, width, height).
```

Процедури анімації Opengl

Растрові операції і функції привласнення коду кольору ***(color-index assignment)*** містяться в кореневій бібліотеці, а процедури зміни значень таблиці кольорів є в ***GLUT***. Інші операції растрової анімації є лише в процедурах ***GLUT***, оскільки вони залежать від використовуваної системи вікон. Крім того, в деяких апаратних системах можуть бути відсутніми особливості комп'ютерної анімації, такі як подвійна буферизація.

Операції подвійної буферизації, якщо вони є, викликаються за допомогою команди ***GLUT***.

```
glutInitDisplayMode(Glut_double);
```

В результаті створюється два буфери, передній і задній, які можна використовувати попеременно для оновлення зображення на екрані. Поки один буфер виступає в ролі буфера оновлення поточного вікна на екрані, в іншому буфері будується наступний кадр анімації. Момент, коли буфери міняються ролями, задається командою

```
glutSwapBuffers();
```

Щоб визначити, чи можлива в системі робота з подвійною буферизацією, можна видати наступний запит.

```
glGetBooleanv(Gl_DOUBLEBUFFER,  status);
```

Значення ***GL_TRUE***, видане у відповідь, привласнюється параметру масиву ***status***, якщо в системі доступні передній і задній буфери. Інакше повертається значення ***GL_FALSE***. Для безперервної анімації також можна використовувати наступну процедуру.

```
glutIdleFunc(animationFcn);
```

Тут параметру ***animationFcn*** привласнюється ім'я процедури, яка послідовно збільшуватиме параметри анімації. Дана процедура безперервно виконується, поки не залишиться подій у вікні дисплея, які потрібно обробити. Щоб блокувати процедуру ***glutIdelFunc***, її аргументу привласнюється значення ***NULL*** або ***0***. Нижче наведений приклад програми анімації, в якій правильний шестикутник безперервно обертається в площині ***xy*** довкола осі ***z***. Початок тривимірних екранних координат розташовується в центрі вікна на екрані, так що вісь ***z*** проходить через цей центр. У процедурі ***Init*** таблиця відображення використовується для завдання описання правильного шестикутника, центр якого спочатку знаходиться в крапці з екранними координатами (150, 150), а радіус (відстань від центру багатокутника до будь-якої вершини) дорівнює 100 . У функції відображення ***displayHex*** задається початковий кут обертання 0° довкола осі ***z*** і викликається процедура ***glutSwapBuffers***. Щоб ініціювати обертання, використовується процедура ***mouseFcn***, яка послідовно збільшує кут повороту на 3° при натисненні середньої кнопки миші. Розрахунок кута, що послідовно збільшується, виконується в процедурі, яка викликається процедурою ***glutIdleFunc*** процедурі ***mouseFcn***. Обертання припиняється натисненням правої кнопки миші, що приводить до виклику процедури ***glutIdleFunc*** з аргументом ***NULL***.

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), вхід до головного циклу обробки подій ***GLUT***.

2) Сформувати тривимірне зображення за допомогою графічних примітивів. Передбачити можливості модельної, видової або проекційної трансформацій

над об’єктом. Передбачити можливість управління (наприклад поворот об’єкта за допомогою клавіш “+” та “-”) об’єктом за допомогою клавіатури (мишки). Зображення знаходиться у таблиці №1, згідно варіанту.

3.1 Розробити та використати функцію по переносу тривимірного зображення. Константа переносу знаходиться у таблиці №1, згідно варіанту.
3.2 Розробити та використати функцію по повороту тривимірного зображення. Кут повороту знаходиться у таблиці №1, згідно варіанту.
3.3 Розробити та використати функцію по переносу тривимірного зображення. Константа масштабування знаходиться у таблиці №1, згідно варіанту. (на оцінку задовільно)
4) Додати код для визначення границь зображення (використовувати перспективну матрицю). (на оцінку добре)
5) Зробити зображення з таблиці 1 анімованим. Завдання анімації див. у таблиці 2-3. (на оцінку відмінно)

Таблиця1.

№	Фігура	Кут повороту	Константа переносу		Константа масштабування	
			Tx	Ty	Sx	Sy
1	Куля	pi/4	50	-50	2	0.5
2	Прзма	-pi/4	0	100	2	0.5
3	Паралелепіед	pi/2	100	100	1	2
4	Сфера	pi	150	-50	0.5	1
5	Чотирикутна піраміда	-pi	-150	50	2	1
6	Призма	-pi/2	50	50	0.5	2
7	Тетраедр	pi	150	0	1	2
8	Куля	pi/4	-100	0	2	1
9	Чотирикутна піраміда	-pi/4	-50	-100	0.5	2
10	Куб	pi/2	0	-100	1	0.5
11	Призма	-pi	100	-150	2	2
12	Чотирикутна піраміда	-pi/2	50	-150	0.5	0.5

13	Тетраедр	pi/6	-150	-50	0.5	3
14	Куля	pi/2	100	50	2	1
15	Сфера	pi/6	50	100	1	2
16	Куб	pi	0	50	2	1
17	Призма	pi/2	75	0	0.5	2
18	Тетраедр	-pi/6	-50	75	1	2
19	Куб	-pi	-100	0	1	0.5
20	Сфера	-pi/4	0	-75	2	3
21	Паралелепіпед	pi/4	-75	150	0.5	1
22	Чотирикутна піраміда	-pi/2	150	100	0.5	0.5
23	Куб	-pi	50	0	2	2
24	Призма	pi	50	-150	2	1
25	Куля	pi/2	100	-100	0.5	0.5
26	Тетраедр	pi/6	0	150	1	0.5
27	Чотирикутна піраміда	-pi/4	-100	50	2	3
28	Тетраедр	-pi/6	75	-100	1	2
29	Сфера	pi/2	-75	0	1	1
30	Куб	pi	-50	75	0.5	1
31	Призма	-pi/6	150	-50	2	3
32	Сфера	pi/3	-150	0	0.5	0.5
33	Куля	pi/6	50	150	1	2
34	Чотирикутна піраміда	-pi/3	0	50	1	3
35	Паралелепіпед	-pi	-75	100	2	1

Таблиця 2

№ варіанту	**Назва малюнку**
1	Октаедр
2	Тор
3	Конус

4	Циліндр
5	Ікосаедр
6	Паралелепіпед
7	Куб
8	Конус
9	Куля
10	Тетраедр
11	Чотирикутна піраміда
12	Конус
13	Куб
14	Сфера
15	Куля
16	Тетраедр
17	Октаедр
18	Ікосаедр
19	Додекаедр
20	Чотирикутна піраміда
21	Сфера
22	Циліндр
23	Конус
24	Куб
25	Призма
26	Октаедр
27	Сфера
28	Призма
29	Паралелепіпед
30	Чотирикутна піраміда
31	Тор
32	Куб
33	Сфера
34	Ікосаедр
35	Конус

Таблиця 3.

№	Завдання анімації
1	Рух 1-ої фігури навколо будь-якого свого ребра, а 2-ої навколо першої фігури
2	Рух 1-ї фігури по квдратній траєкторії, а 2-ї навколо своєї осі
3	Рух фігур навколо осі однієї з них в різні сторони (фігури можуть бути розміщені одна над одною)
4	Рух обох тіл по замкнених лініях (різних одна від одної)
5	Зобразити хаотичний рух для 1-ї фігури, для 2-ї обертання наволо вісі координат Z
6	Рух 1-ої фігури по колу, а 2-ої по еліпсу
7	Обертання 1-ої фігури навколо своєї вершини, а 2-ої навколо одного з ребер нижньої основи
8	1-ша фігура обертається навколо 2-ої, а 2-га навколо свого центру або вершини
9	2-га фігура імітує стрибки, а 1-ша рух навколо довільної осі
10	Рух кулі та сфери відносно будь-якої осі
11	Рух 1-ої фігури навколо будь-якого свого ребра, а 2-ої навколо першої фігури
12	Рух фігур на зустріч одна одній
13	Рух 1-ї фігури навколо своєї осі, 2-ї вертикально до низу
14	Рух 1-ої фігури по колу, а 2-ої по еліпсу
15	Рух кулі та сфери відносно будь-яких осей координат
16	Рух 1-ої фігури з лівого верхнього кута - в правий нижній, а 2-ої з правого верхнього в лівий нижній
17	Рух 1-ої фігури навколо своєї осі, а рух 2-ої навколо першої фігури
18	Рух фігур навколо своїх осей
19	1-ю фігурою відобразити горизонтальний рух, 2-ю вертикальний
20	Рух 1-ох фігури по замкненій лінії (обов'язково у вигляді вісімки), а рух 2-ої проходить навколо вершини
21	Рух фігур на зустріч одна одній
22	Переміщення фігур по діагоналі
23	2-га фігура імітує стрибки, а 1-ша рух навколо довільної осі
24	Рух по будь-якій замкненій ламаній (не менше 4-х вершин)
25	Рух 1-ї фігури по квдратній траєкторії, а 2-ї навколо своєї осі

26	Рух по будь-якій замкненій ламаній (не менше 4-х вершин для кожної фігури)
27	Обертання 1-ої фігури навколо своєї вершини, а 2-ої навколо одного з ребер нижньої основи
28	1-ша фігура обертається навколо 2-ої, а 2-га навколо свого центру або вершини
29	Переміщення фігур по діагоналі
30	Рух 1-ї фігури навколо своєї осі, 2-ї вертикально до низу
31	1-ю фігурою відобразити горизонтальний рух, 2-ю вертикальний
32	Рух 1-ої фігури навколо своєї осі, а рух 2-ої навколо першої фігури
33	Рух кулі та сфери відносно будь-якої осі
34	Переміщення фігур по діагоналі
35	Рух фігур навколо своїх осей