```cpp
#include <bits/stdc++.h>
#include <GL/glut.h>

#define GL_GLEXT_PROTOTYPES

using namespace std;

double rotate_y = 45;
double rotate_x = 45;

vector<vector<double>> v{{0.5, -0.5, -0.5, 1}, {0.5, 0.5, -0.5, 1},
{-0.5, 0.5, -0.5, 1}, {-0.5, -0.5, -0.5, 1}, {0.5, -0.5, 0.5, 1},
{0.5, 0.5, 0.5, 1}, {-0.5, 0.5, 0.5, 1}, {-0.5, -0.5, 0.5, 1}};
vector<vector<double>> r = v;
vector<vector<double>> z{{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0},
{0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 0,
0}};
vector<vector<double>> rx{{0, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0},
{0, 0, 0, 1}};
vector<vector<double>> ry{{1, 0, 0, 0}, {0, 0, 0, 0}, {0, 0, 1, 0},
{0, 0, 0, 1}};
vector<vector<double>> rz{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 0},
{0, 0, 0, 1}};
vector<vector<double>> cavalier{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 1,
1, 0}, {0, 0, 0, 1}};
vector<vector<double>> cabinet{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0,
0.5, 0}, {0, 0, 0, 1}};
vector<vector<double>> temp;

void initGL()
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClearDepth(1.0f);
    glPointSize(5);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glShadeModel(GL_SMOOTH);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    gluOrtho2D(-50, 50, -50, 50);
}

void matrix_multiply(vector<vector<double>> &vr)
{
    temp = v;
    v = z;
    for (int i = 0; i < v.size(); ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            for (int k = 0; k < 4; ++k)
            {
                v[i][j] += temp[i][k] * vr[k][j];
            }
        }
    }
}

// Function for parallel projection
void parallel(vector<vector<double>> &vr, double x0, double y0,
double z0, double n1, double n2, double n3, double a, double b,
double c)
{
    double d0 = n1 * x0 + n2 * y0 + n3 * z0;
    double d1 = n1 * a + n2 * b + n3 * c;
    vector<vector<double>> transform = {{d1 - a * n1, -b * n1, -c *
n1, 0}, {-a * n2, d1 - b * n2, c * n2, 0}, {-a * n3, -b * n3, d1 - c
* n3, 0}, {a * d0, b * d0, c * d0, d1}};
    matrix_multiply(transform);
}

// Function for perspective projection
void perspective(vector<vector<double>> &vr, double x0, double y0,
double z0, double n1, double n2, double n3, double a, double b,
double c)
{
    double d0 = n1 * x0 + n2 * y0 + n3 * z0;
    double d1 = n1 * a + n2 * b + n3 * c;
    double d = d0 - d1;
    vector<vector<double>> transform = {{n1 * a + d, b * n1, c * n1,
n1}, {a * d1, b * n2 + d, c * n2, n2}, {a * n3, b * n3, c * n3 + d,
n3}, {-a * d0, -b * d0, -c * d0, -d1}};
    matrix_multiply(transform);
}

// Function for isometric axonometric parallel projection
void isometric()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(3), n2 = 1 /
sqrt(3), n3 = 1 / sqrt(3), a = 1 / sqrt(3), b = 1 / sqrt(3), c = 1 /
sqrt(3);
    parallel(v, x0, y0, z0, n1, n2, n3, a, b, c);
```

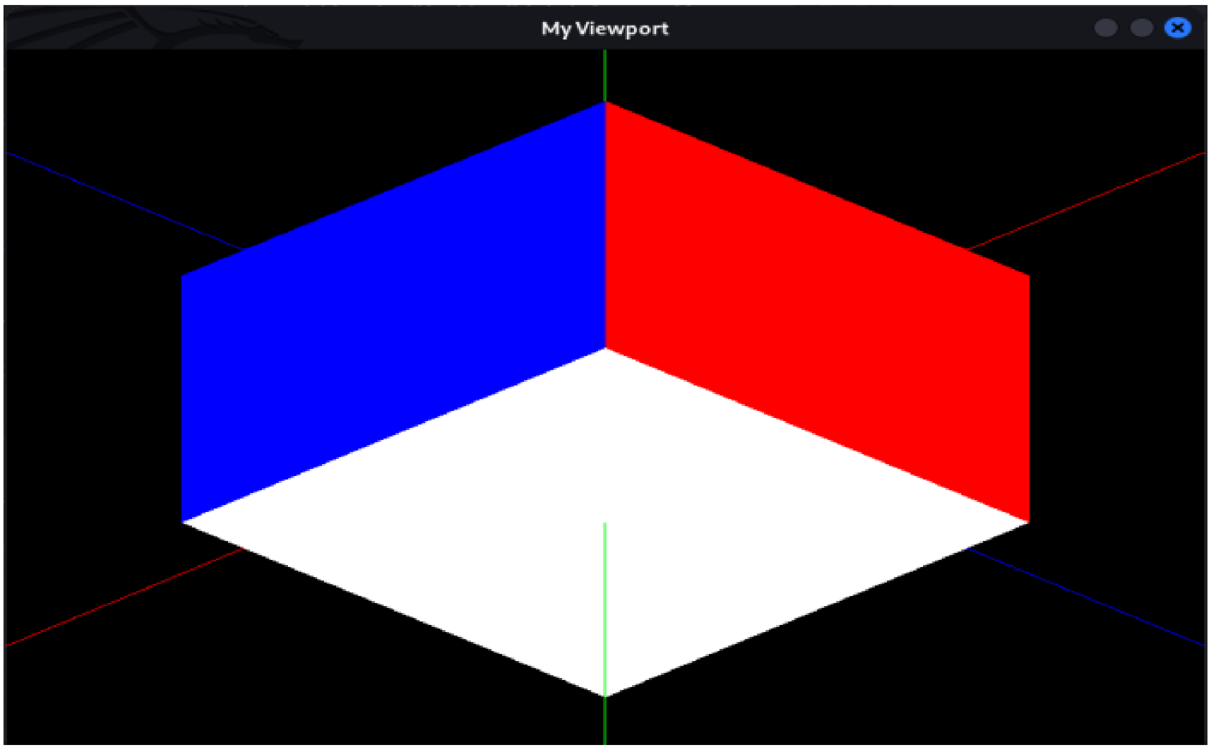
```
}

// Function for diametric axonometric parallel projection
void diametric()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(6), n2 = 2 /
sqrt(6), n3 = 1 / sqrt(6), a = 1 / sqrt(6), b = 2 / sqrt(6), c = 1 /
sqrt(6);
    parallel(v, x0, y0, z0, n1, n2, n3, a, b, c);
}

// Function for trimetric axonometric parallel projection
void trimetric()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(13), n2 = 2 /
sqrt(13), n3 = 3 / sqrt(13), a = 1 / sqrt(13), b = 2 / sqrt(13), c =
3 / sqrt(13);
    parallel(v, x0, y0, z0, n1, n2, n3, a, b, c);
}

// Function for one point perspective projection
void onepoint()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(3), n2 = 1 /
sqrt(3), n3 = 1 / sqrt(3), a = 0, b = 0, c = 1 / sqrt(3);
    perspective(v, x0, y0, z0, n1, n2, n3, a, b, c);
}

// Function for two point perspective projection
void twopoint()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(6), n2 = 1 /
sqrt(6), n3 = 2 / sqrt(6), a = 1 / sqrt(6), b = 1 / sqrt(6), c = 0;
    perspective(v, x0, y0, z0, n1, n2, n3, a, b, c);
}

// Function for three point perspective projection
void threepoint()
{
    v = r;
    double x0 = 1, y0 = 0, z0 = 0, n1 = 1 / sqrt(13), n2 = 2 /
sqrt(13), n3 = 3 / sqrt(13), a = 1 / sqrt(13), b = 2 / sqrt(13), c =
3 / sqrt(13);
    perspective(v, x0, y0, z0, n1, n2, n3, a, b, c);
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'x': // Finding x projection
        v = r;
        matrix_multiply(rx);
        break;
    case 'y': // Finding y projection
        v = r;
        matrix_multiply(ry);
        break;
    case 'z': // Finding z projection
        v = r;
        matrix_multiply(rz);
        break;
    case 'i': // Finding isometric projection
        v = r;
        isometric();
        break;
    case 'd': // Finding diametric projection
        v = r;
        diametric();
        break;
    case 't': // Finding trimetric projection
        v = r;
        trimetric();
        break;
    case '1': // Finding one point perspective projection
        v = r;
        onepoint();
        break;
    case '2': // Finding two point perspective projection
        v = r;
        twopoint();
        break;
    case '3': // Finding three point perspective projection
        v = r;
        threepoint();
        break;
    case 'r': // Revealing original cube
        rotate_x = 45;
        rotate_y = 45;
        v = r;
        break;
    case 'b': // Finding cabinet parallel projection
        v = r;
        matrix_multiply(cabinet);
        break;
    case 'v': // Finding cavalier parallel projection
        v = r;
        matrix_multiply(cavalier);
        break;
    case 'q': // Exiting the display
        exit(1);
    }
    glutPostRedisplay();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3f(1.0, 1.0, 1.0);
    glRotatef(rotate_x, 1.0, 0.0, 0.0);
    glRotatef(rotate_y, 0.0, 1.0, 0.0);

    // Displaying X-axis
    glBegin(GL_LINE_LOOP);
    glColor3f(1, 0, 0);
    for (int i = -10000; i < 10000; i++)
    {
        glVertex3f(i, 0, 0);
    }
    glEnd();

    // Displaying Y-axis
    glBegin(GL_LINE_LOOP);
    glColor3f(0, 1, 0);
    for (int i = -10000; i < 10000; i++)
    {
        glVertex3f(0, i, 0);
    }
    glEnd();

    // Displaying Z-axis
```

```
glBegin(GL_LINE_LOOP);
glColor3f(0, 0, 1);
for (int i = -10000; i <= 10000; i++)
{
    glVertex3f(0, 0, i);
}
glEnd();

glBegin(GL_QUADS);

// front side - blue
glColor3f(0.0, 0.0, 1.0);
glVertex3f(v[0][0], v[0][1], v[0][2]);
glVertex3f(v[1][0], v[1][1], v[1][2]);
glVertex3f(v[2][0], v[2][1], v[2][2]);
glVertex3f(v[3][0], v[3][1], v[3][2]);

// back side - green
glColor3f(0.0, 1.0, 0.0);
glVertex3f(v[4][0], v[4][1], v[4][2]);
glVertex3f(v[5][0], v[5][1], v[5][2]);
glVertex3f(v[6][0], v[6][1], v[6][2]);
glVertex3f(v[7][0], v[7][1], v[7][2]);

// right side - red
glColor3f(1.0, 0.0, 0.0);
glVertex3f(v[0][0], v[0][1], v[0][2]);
glVertex3f(v[1][0], v[1][1], v[1][2]);
glVertex3f(v[5][0], v[5][1], v[5][2]);
glVertex3f(v[4][0], v[4][1], v[4][2]);

// left side - purple
glColor3f(1.0, 0.0, 1.0);
glVertex3f(v[7][0], v[7][1], v[7][2]);
glVertex3f(v[6][0], v[6][1], v[6][2]);
glVertex3f(v[2][0], v[2][1], v[2][2]);
glVertex3f(v[3][0], v[3][1], v[3][2]);

// top side - yellow
glColor3f(1.0, 1.0, 0.0);
glVertex3f(v[5][0], v[5][1], v[5][2]);
glVertex3f(v[1][0], v[1][1], v[1][2]);
glVertex3f(v[2][0], v[2][1], v[2][2]);
glVertex3f(v[6][0], v[6][1], v[6][2]);

// bottom side - white
```

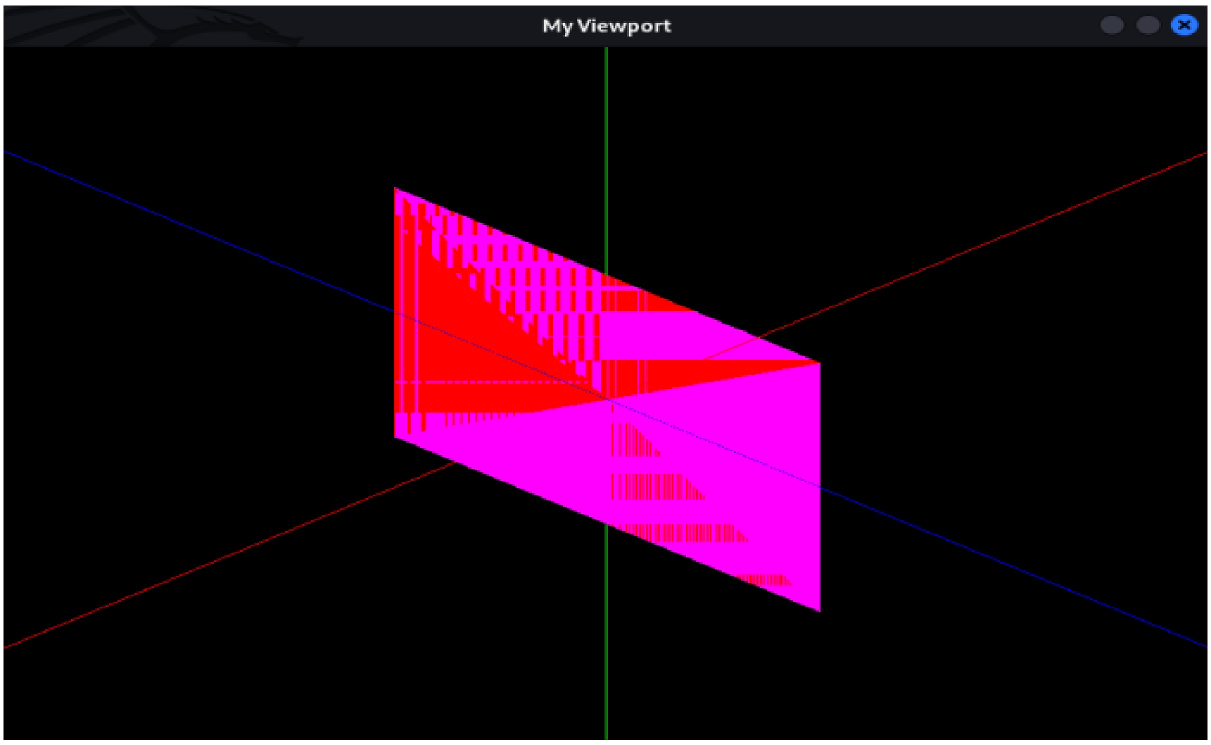
```
    glColor3f(1.0, 1.0, 1.0);
    glVertex3f(v[0][0], v[0][1], v[0][2]);
    glVertex3f(v[4][0], v[4][1], v[4][2]);
    glVertex3f(v[7][0], v[7][1], v[7][2]);
    glVertex3f(v[3][0], v[3][1], v[3][2]);

    glEnd();
    glFlush();
    glutSwapBuffers();
}

// Rotating the cube using arrow keys
void specialKeys(int key, int x, int y)
{
    if (key == GLUT_KEY_RIGHT)
        rotate_y += 5;

    else if (key == GLUT_KEY_LEFT)
        rotate_y -= 5;

    else if (key == GLUT_KEY_UP)
        rotate_x += 5;

    else if (key == GLUT_KEY_DOWN)
        rotate_x -= 5;

    glutPostRedisplay();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("My Viewport");
    glEnable(GL_DEPTH_TEST);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutSpecialFunc(specialKeys);
    initGL();
    glutMainLoop();
    return 0;
}
```

Output:

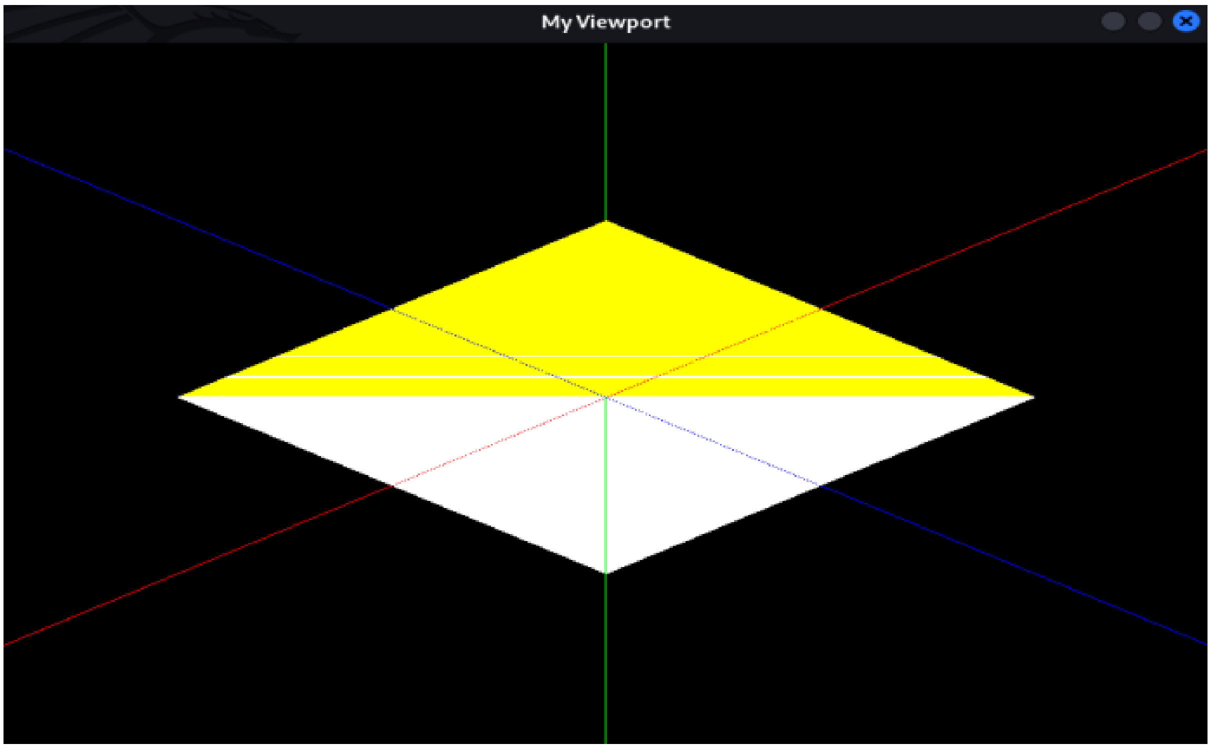
Original Object (Default, key=r):



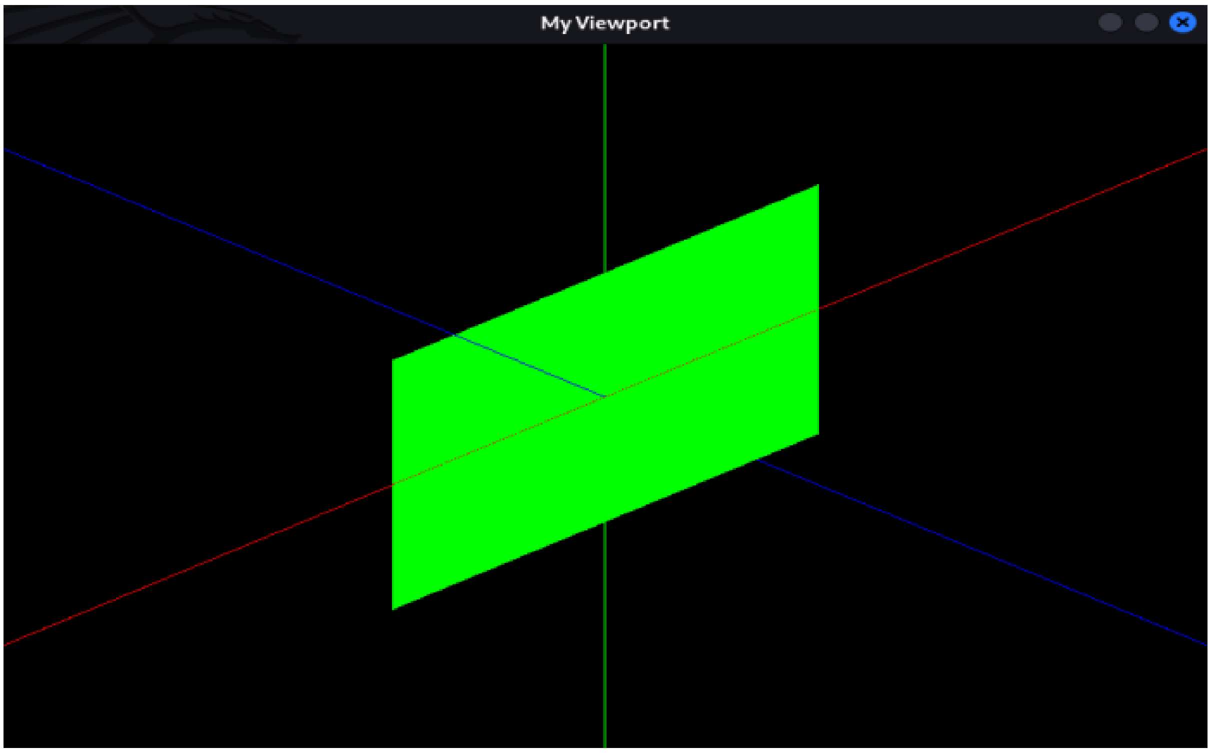
Orthographic projection on x=0 (key=x):



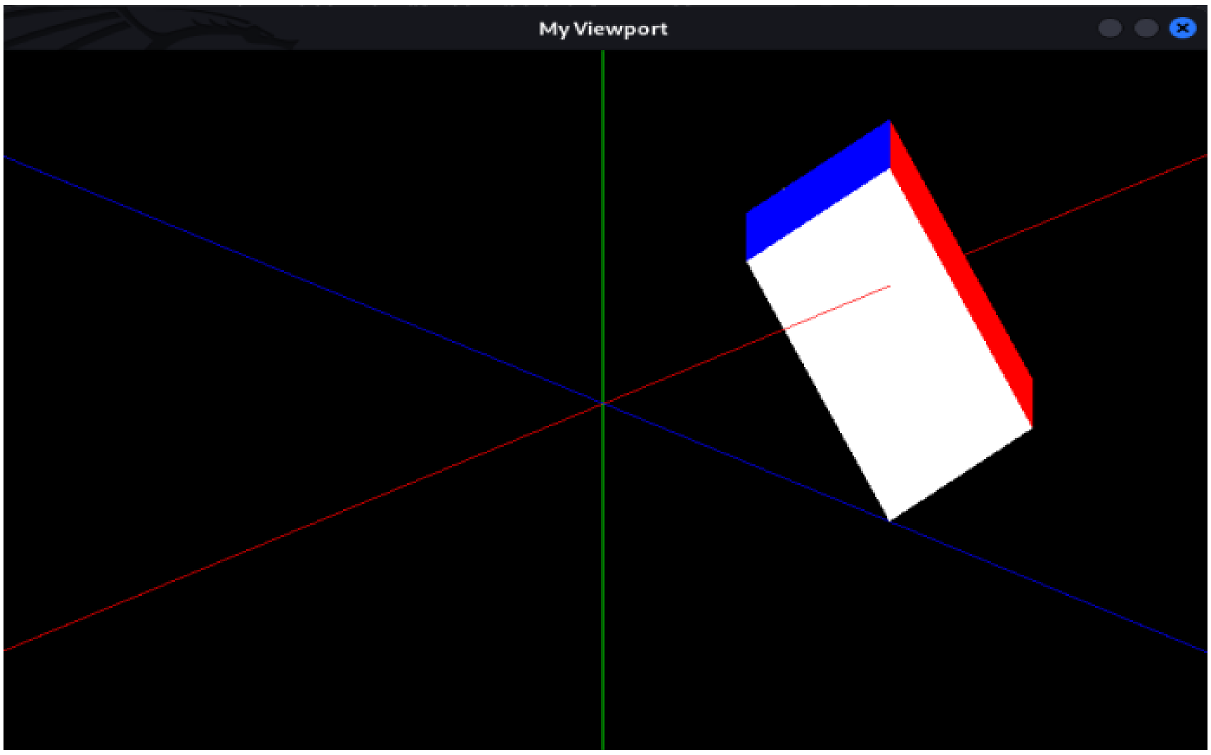
Orthographic projection on y=0 (key=y):



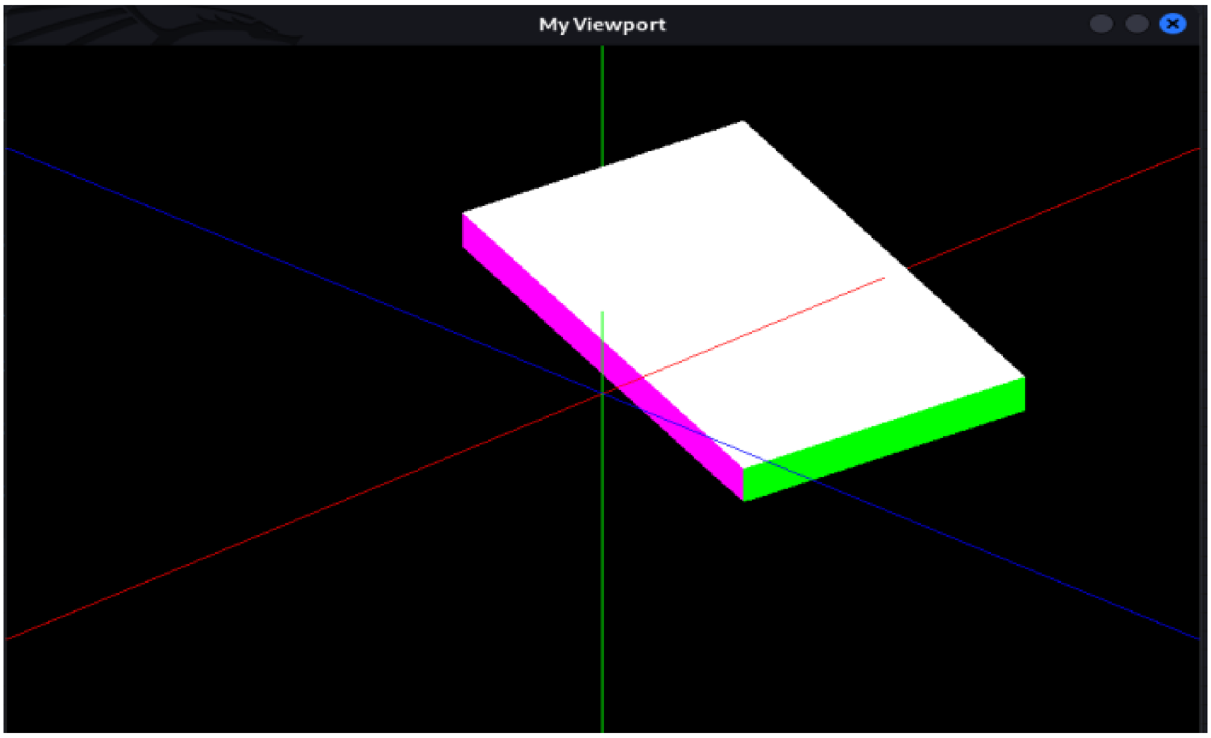
Orthographic projection on z=0 (key=z):



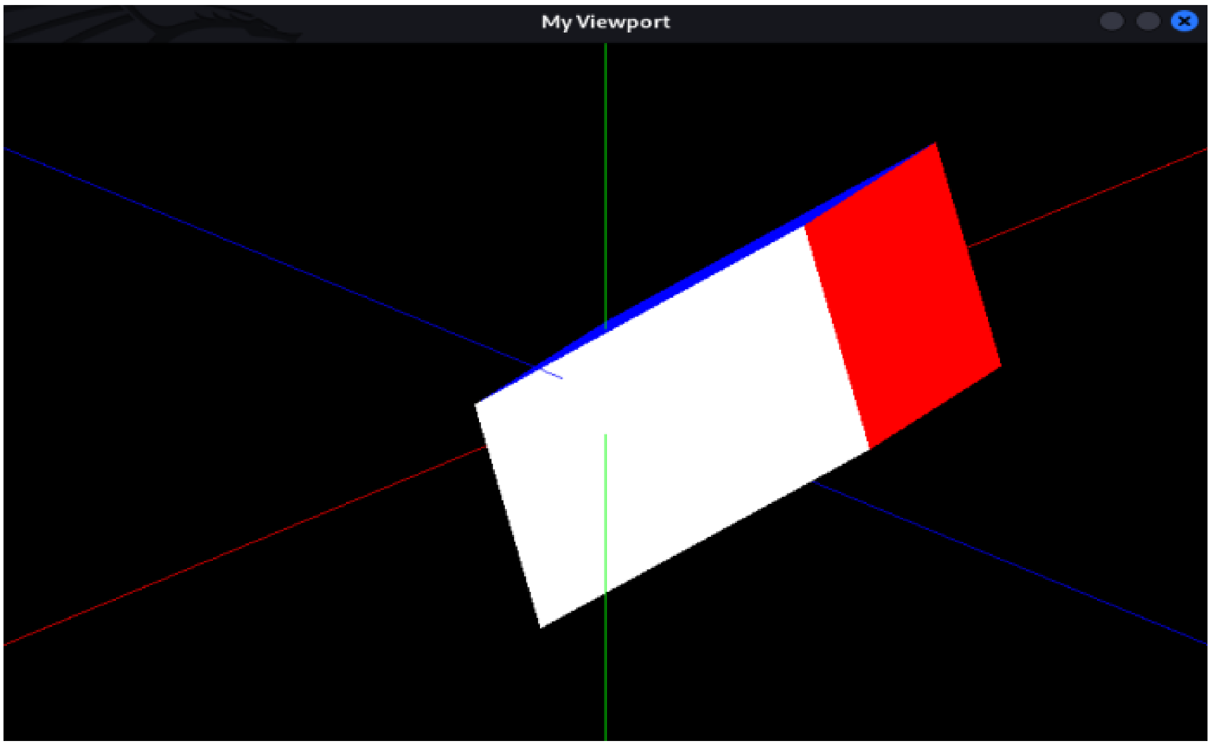
Isometric projection (key=i):



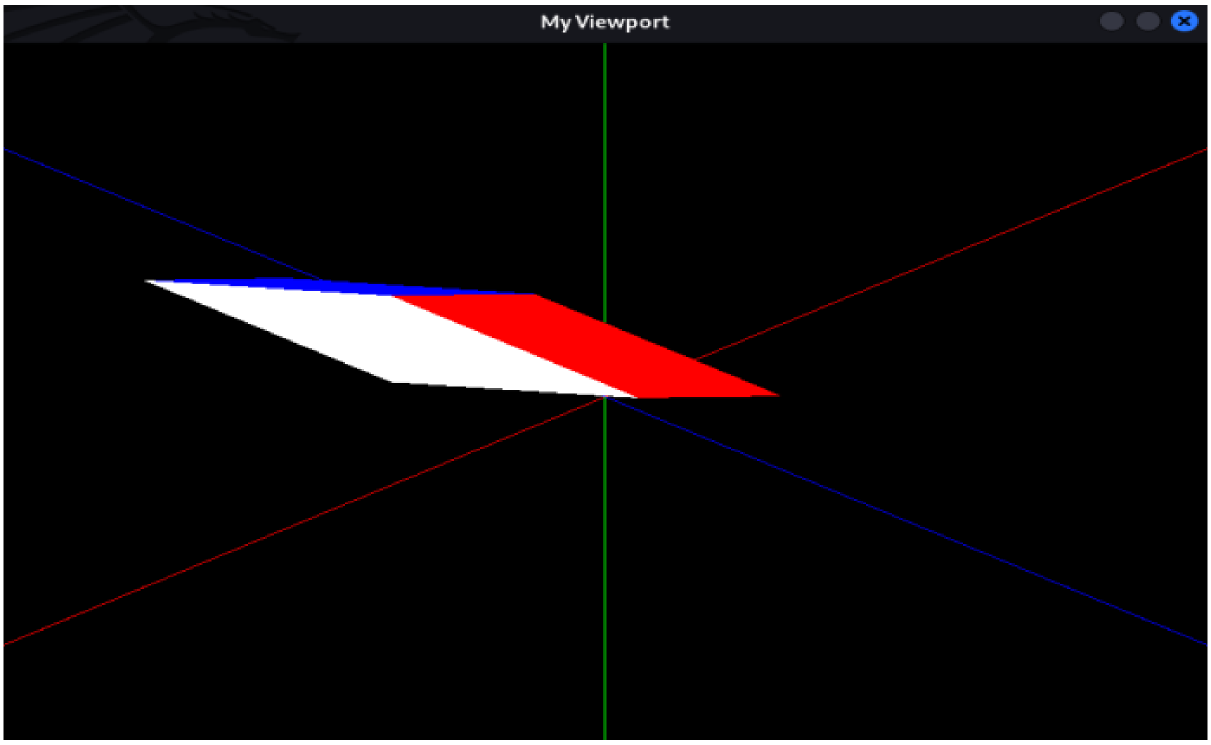
Diametric projection (key=d):



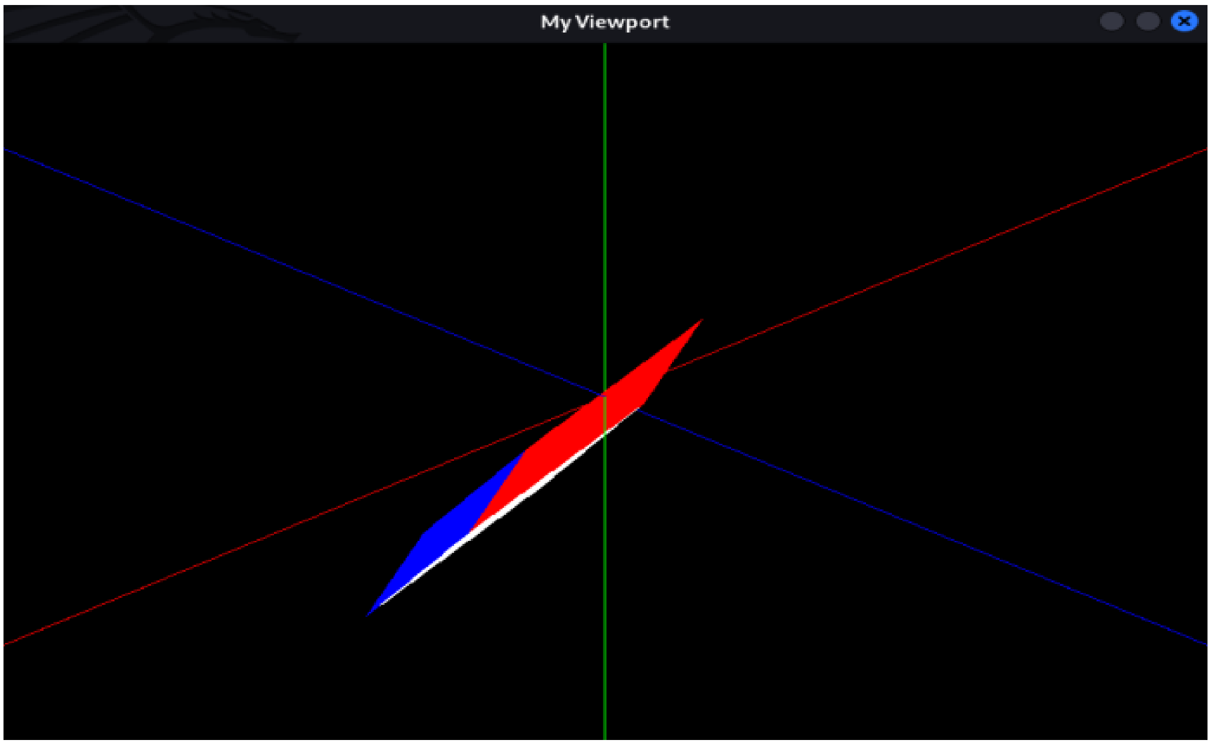
Trimetric projection (key=t):



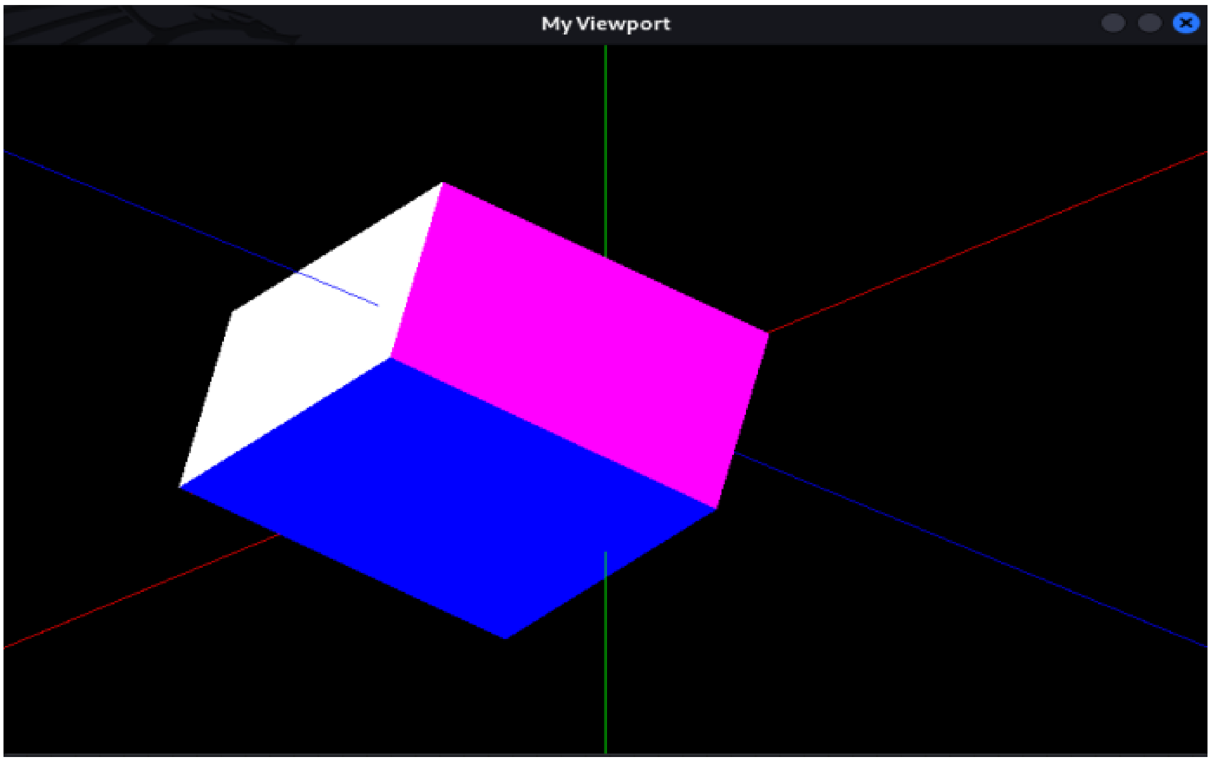
One point perspective (key=1):



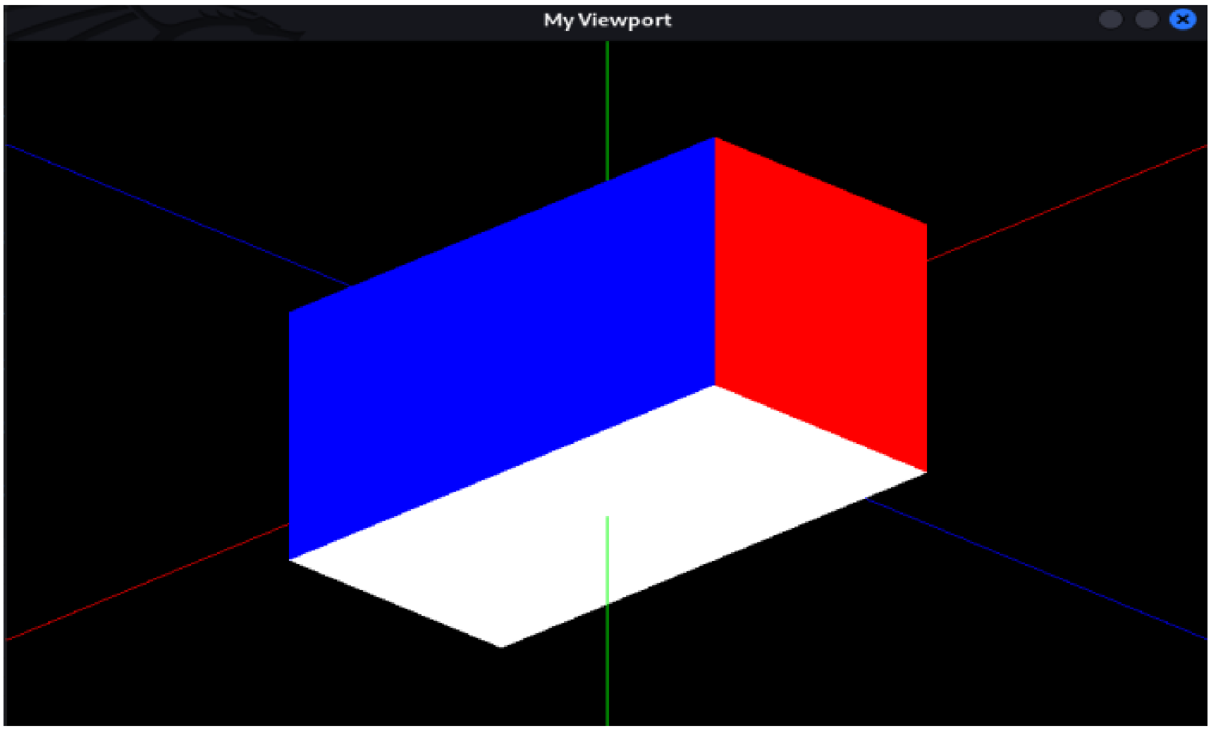
Two point perspective (key=2):



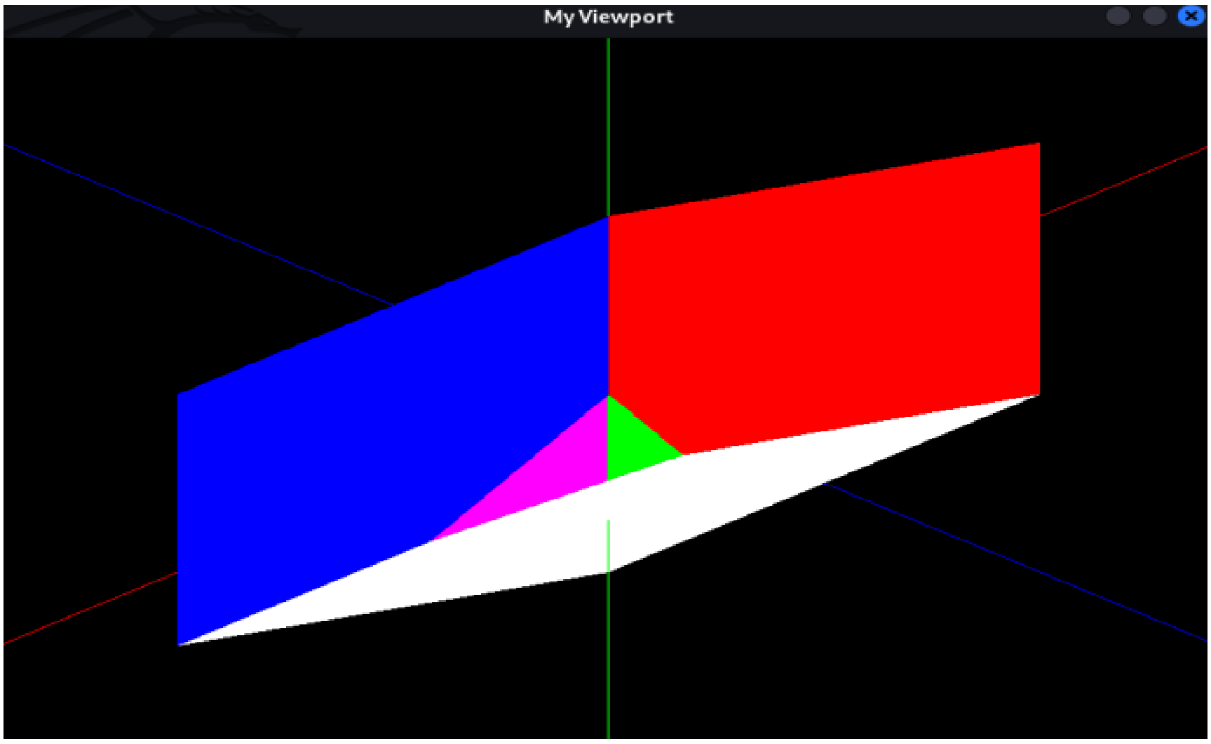
Three point perspective (key=3):



Cabinet parallel projection (key=b):



Cavalier parallel projection (key=v):


My Viewport