**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ВТ**

отчет

**по лабораторной работе №3**

**по дисциплине «Компьютерная графика»**

**Тема:** **Формирования различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость при ее визуализации (выводе на экран дисплея)**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 6306 |  | Гордиенко М. Е. |
|  |  | Пустовойтова А. А. |
| Преподаватель |  | Матвеева И. В. |

Санкт-Петербург

2019

**Цель работы.**

Исследовать методы формирования различных поверхностей с помощью их пространственного разворота и ортогонального проецирования на плоскость при визуализации.

**Задание.**

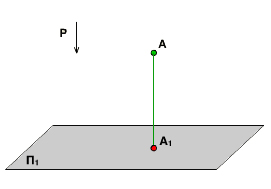
Сформировать билинейную поверхность на основе произвольного задания ее четырёх угловых точек. Обеспечить ее поворот относительно осей *X* и *Y*.

**Теория.**

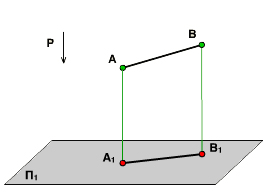
Ортогональное проецирование — это частный случай параллельного проецирования. При ортогональном проецировании проецирующие лучи перпендикулярны к плоскости проекций.

Аппарат такого проецирования состоит из одной плоскости проекций.

Чтобы получить ортогональную проекцию точки А, через неё надо провести проецирующий луч перпендикулярно к П1. Точка А1 называется ортогональной или прямоугольной проекцией точки А.



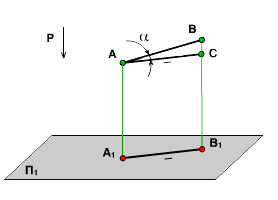
Чтобы получить ортогональную проекцию **А1В1** отрезка **АВ**, на плоскость **П1**, необходимо через точки **А** и **В** провести проецирующие прямые, перпендикулярные **П1**. При пересечении проецирующих прямых с плоскостью **П1** получатся ортогональные проекции **А1** и **В1** точек **А** и **В**. Соединив ортогональные проекции **А1** и **В1** получим ортогональную проекцию **А1В1** отрезка **АВ**.



Все свойства параллельного проецирования выполнимы и для ортогонального проецирования. Однако ортогональные проекции обладают ещё некоторыми свойствами.

**Свойства ортогонального проецирования:**  
1. Длина отрезка равна длине его проекции, делённой на косинус угла наклона отрезка к плоскости проекций.

Возьмём прямую **АВ** и построим её ортогональную проекцию **А1В1** на плоскость **П1**. Если провести прямую **АС || А1В1**, то из треугольника **АВС** следует, что **|АС| : |АВ| = cos a** или **|АВ| = |А1В1| : cos a**, т. к. **|А1В1| = |АС|**.

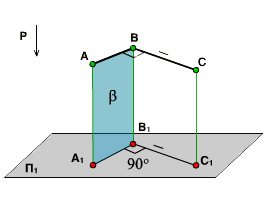


2. Кроме того, для ортогонального проецирования будет справедлива **теорема о проецировании прямого угла:**

**Теорема:**  Если хотя бы одна сторона прямого угла параллельна плоскости проекций, а вторая ей не перпендикулярна, то угол на эту плоскость проецируется в натуральную величину.

**Доказательство:**

Дан прямой угол **АВС**, у которого по условию прямая **ВС http://reshal.ru/wp-content/uploads/2011/12/per.gifАВ** и **ВС ||** плоскости проекций **П1**. По построению прямая **ВС** **http://reshal.ru/wp-content/uploads/2011/12/per.gif**к проецирующему лучу **ВВ1**. Следовательно, прямая **ВС** **http://reshal.ru/wp-content/uploads/2011/12/per.gif**к плоскости **b (АВхВВ1)**, т. к. она **http://reshal.ru/wp-content/uploads/2011/12/per.gif**к двум пересекающимся прямым , лежащим в этой плоскости. По условию прямая **В1С1 || ВС**, поэтому тоже **http://reshal.ru/wp-content/uploads/2011/12/per.gif**к плоскости **b**, т. е. и прямой **А1В1** этой плоскости. Следовательно, угол между прямыми **А1В1** и **В1С1** равен 90°, что и требовалось доказать.

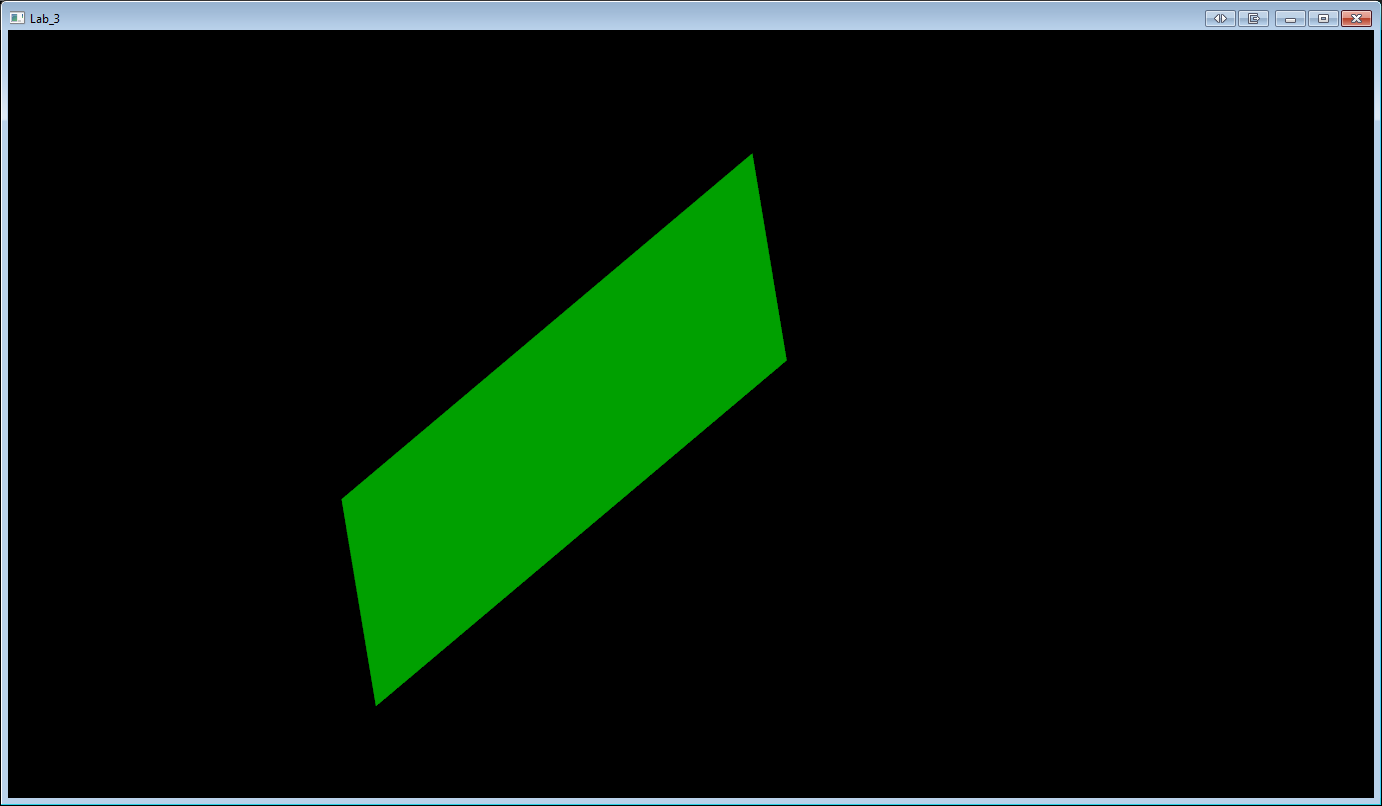


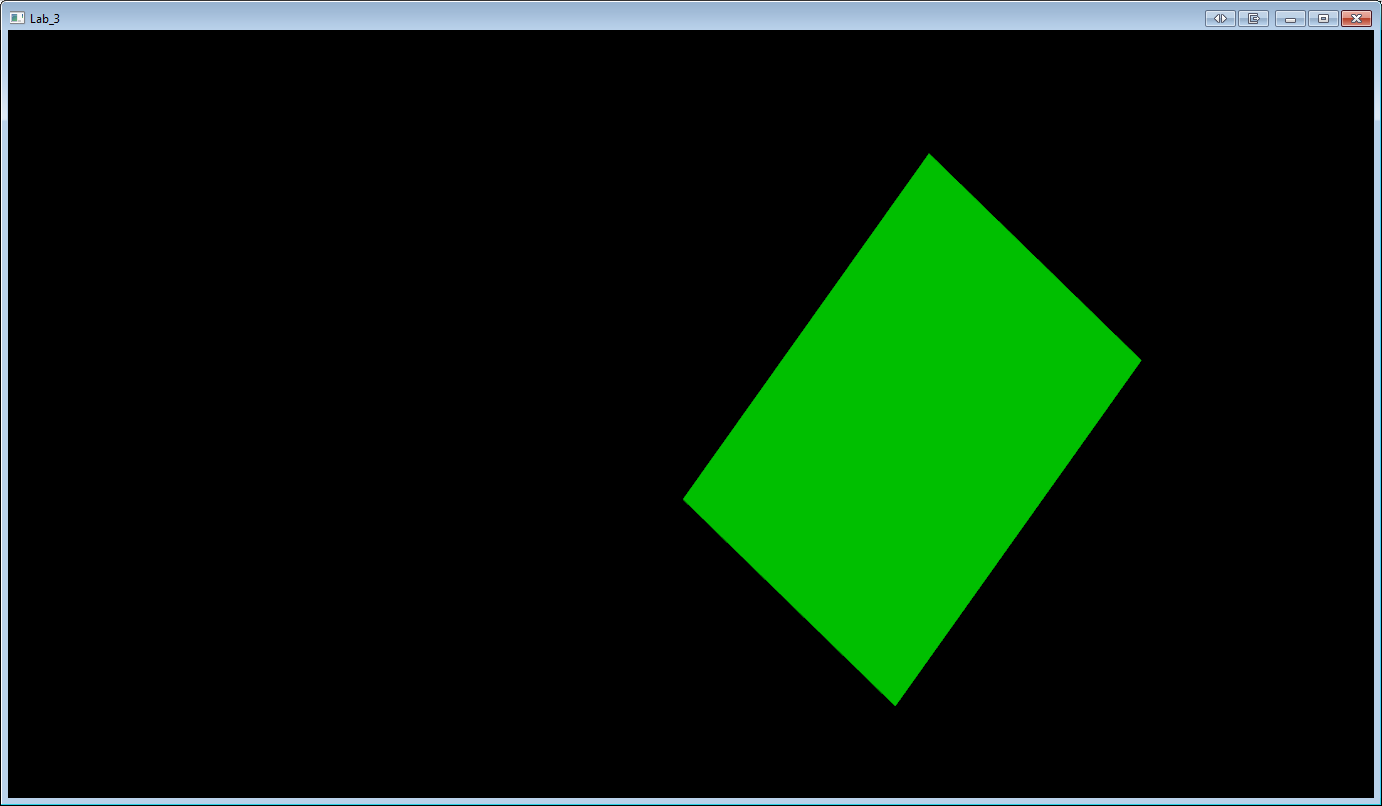
Ортогональное проецирование обеспечивает простоту геометрических построений при определении ортогональных проекций точек, а так же возможность сохранять на проекциях форму и размеры проецируемой фигуры. Эти достоинства обеспечили ортогональному проецированию широкое применение в техническом черчении.

Рассмотренные методы проецирования позволяют решить прямую задачу начертательной геометрии, т. е. по оригиналу построить плоский чертёж. Полученные таким образом проекции на одну плоскость дают неполное представление о предмете, его форме и положении в пространстве, т. е. такой чертёж не обладает свойством обратимости.

**Описание реализации.**

На экран выводится произвольная плоскость, заданная четырьмя угловыми точками, после чего её поворот относительно осей X и Y осуществляется клавишами со стрелками на клавиатуре.







(На приведённых выше рисунках изображена одна и та же плоскость, повёрнутая на различные углы относительно осей X и Y)

**Вывод.**

В ходе лабораторной работы были исследованы формирования различных плоскостей с использованием ортогонального проектирования на плоскость визуализации.

**Приложение.**

**Код программы.**

Метод отрисовки плоскости:

void FourPointLinearPlane::Draw(sf::RenderWindow\* window, sf::Color color, sf::Color color2) {

sf::ConvexShape shape;

shape.setPointCount(4);

float cos = norm.z / (sqrt(pow(norm.x, 2) + pow(norm.y, 2) + pow(norm.z, 2)));

if (cos < 0)

shape.setFillColor(sf::Color(color.r - (color.r \* (1 - abs(cos)) / 1.25f), color.g - (color.g \* (1 - abs(cos)) / 1.25f), color.b - (color.b \* (1 - abs(cos)) / 1.25f)));

else shape.setFillColor(sf::Color(color2.r - (color2.r \* (1 - abs(cos)) / 1.25f), color2.g - (color2.g \* (1 - abs(cos)) / 1.25f), color2.b - (color2.b \* (1 - abs(cos)) / 1.25f)));

shape.setPoint(0, points(0, 0).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(1, points(0, 1).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(2, points(1, 1).Isometric(window->getSize().x, window->getSize().y));

shape.setPoint(3, points(1, 0).Isometric(window->getSize().x, window->getSize().y));

window->draw(shape);

}

sf::Vector2f Point::Isometric(float xSize, float ySize) {

return sf::Vector2f(pos.x + xSize / 2, ySize / 2 - pos.y);

}

Методы поворота точек относительно осей:

void Point::RotateX(float angl) {

vector<float> posV(3);

posV(0) = pos.x; posV(1) = pos.y; posV(2) = pos.z;

matrix<float> rotateM(3, 3);

rotateM(0, 0) = 1; rotateM(0, 1) = 0; rotateM(0, 2) = 0;

rotateM(1, 0) = 0; rotateM(1, 1) = cos(angl \* M\_PI / 180); rotateM(1, 2) = sin(angl \* M\_PI / 180);

rotateM(2, 0) = 0; rotateM(2, 1) = -sin(angl \* M\_PI / 180); rotateM(2, 2) = cos(angl \* M\_PI / 180);

posV = prod(posV, rotateM);

pos.x = posV(0); pos.y = posV(1); pos.z = posV(2);

}

void Point::RotateY(float angl) {

vector<float> posV(3);

posV(0) = pos.x; posV(1) = pos.y; posV(2) = pos.z;

matrix<float> rotateM(3, 3);

rotateM(0, 0) = cos(angl \* M\_PI / 180); rotateM(0, 1) = 0; rotateM(0, 2) = -sin(angl \* M\_PI / 180);

rotateM(1, 0) = 0; rotateM(1, 1) = 1; rotateM(1, 2) = 0;

rotateM(2, 0) = sin(angl \* M\_PI / 180); rotateM(2, 1) = 0; rotateM(2, 2) = cos(angl \* M\_PI / 180);

posV = prod(posV, rotateM);

pos.x = posV(0); pos.y = posV(1); pos.z = posV(2);

}