Практическое занятие №6

Тема: Трехмерная графика в WPF

Координаты трехмерного пространства

Начало системы координат WPF для графики 2-D отсчитывается от левого верхнего угла области рисования (обычно экрана). В системе 2-D положительные значения оси x откладываются вправо, а положительные значения оси y – сверху вниз. Однако в системе координат 3-D начало располагается в центре отрисовываемой области, положительные значения оси x откладываются вправо, оси y – снизу-вверх, а оси z – из центра к наблюдателю.

Пространство, определяемое этими осями, является стационарной системой отсчета координат для объектов 3-D в приложении WPF. При построении моделей в этом пространстве и создании источников света и камер для их отображения необходимо отличать стационарную систему отсчета координат (или «мировую систему координат») от локальной системы отсчета, которая создается для каждой модели при применении к ней преобразований. Помните, что в зависимости от настройки освещения и камеры, объекты в мировой системе координат могут выглядеть совсем по-другому или вообще быть невидимыми, но положение камеры не изменяет расположения объектов в мировой системе координат.

Камеры и проекции

Разработчики, работающие в координатах 2-D, привыкли к размещению графических примитивов на двухмерном экране. При создании сцены 3-D важно помнить, что фактически создается представление 2-D объектов 3-D. Поскольку сцена 3-D выглядит по-разному в зависимости от точки наблюдения, необходимо указать эту точку наблюдения. Указать эту точку наблюдения для сцены 3-D позволяет класс Camera.

Другой способ понимания того, как представляется сцена 3-D на поверхности 2-D, – это описание сцены как проекции на поверхность просмотра. Камера ProjectionCamera позволяет указать различные проекции и их свойства для изменения того, как наблюдатель видит модели 3-D. Камера PerspectiveCamera указывает проекцию сцены в перспективе. Другими словами, камера PerspectiveCamera предоставляет точку схода перспективы. Можно указать положение камеры в пространстве координат сцены, направление и поле зрения камеры и вектор, определяющий направление «вверх» в сцене. Следующая схема иллюстрирует проекции PerspectiveCamera.

Свойства NearPlaneDistance и FarPlaneDistance камеры ProjectionCamera ограничивают диапазон проекции камеры. Поскольку камеры могут быть расположены в любом месте сцены, фактически можно расположить камеру внутри модели или очень близко от нее, что усложняет правильное распознавание объекта. Свойство NearPlaneDistance позволяет определить минимальное расстояние от камеры, за которым не будут располагаться объекты. И наоборот, свойство FarPlaneDistance позволяет задать расстояние от камеры, дальше которого объекты не будут нарисованы; это гарантирует, что объекты, расположенные слишком далеко для распознавания, не будут включены в сцену.

Освещение

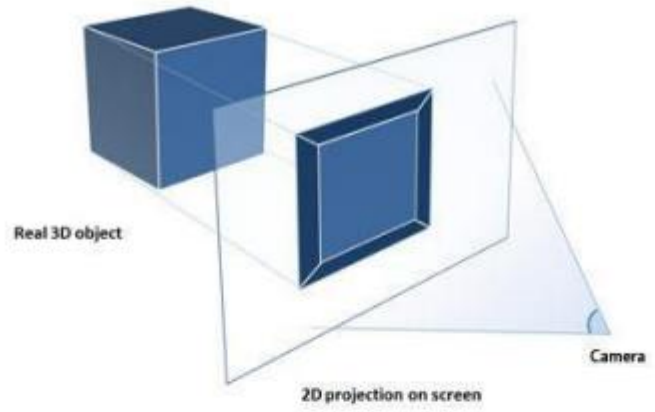
Источники света в графике 3-D выполняют ту же роль, что и реальные источники света: они делают поверхности видимыми. Более того, источники света определяют, какая часть сцены будет включена в проекцию. Объекты источников света в приложении WPF создают различные эффекты света и тени. Они смоделированы на основе поведения различных реальных источников света. Сцена должна включать, как минимум, один источник света, иначе модели будут невидимыми.

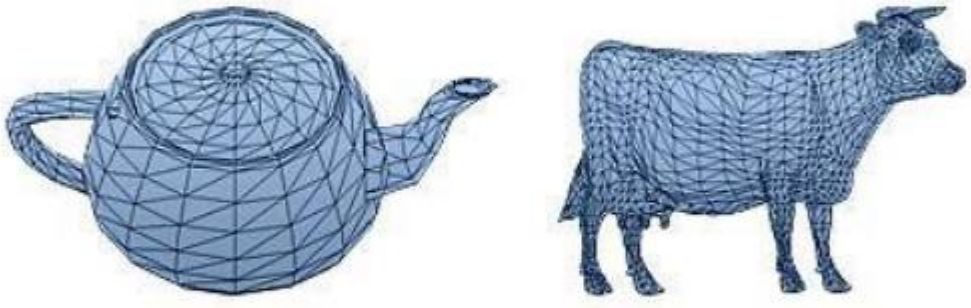
Указанные ниже источники света являются производными от базового класса Light:

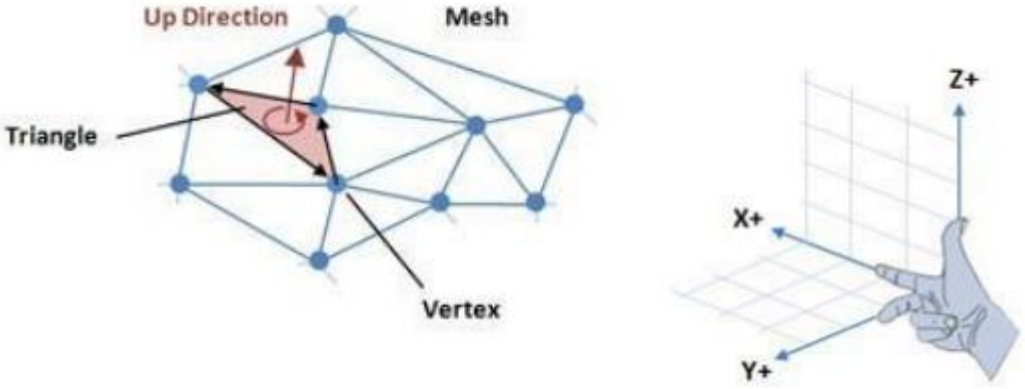
- AmbientLight: создает рассеянное освещение, при котором все объекты освещены одинаково, независимо от их расположения или ориентации.
- DirectionalLight: создает освещение, аналогичное удаленному источнику света. Направленные источники света имеют свойство Direction, которое указывается как объект Vector3D, но без заданного местоположения.
- PointLight создает освещение, как от ближнего источника света. Источники света «PointLights» занимают определенное положение и испускают свет из этого положения. Объекты на сцене освещаются в зависимости от их положения и расстояния относительно источника света. PointLightBase предоставляет свойство Range, которое определяет расстояние, за пределами которого модели не будут освещены светом. Класс «PointLight» также предоставляет свойства затухания, определяющие интенсивность ослабления источника света в зависимости от расстояния. Можно указать константу, линейную или квадратичную интерполяцию затухания источника света.

Исключение SpotLight наследуется от исключения PointLight. Источники света «Spotlights» освещают сцену подобно источникам света PointLight и также имеют расположение и направление. Они проектируют свет в конусообразную область, задаваемую свойствами InnerConeAngleOuterConeAngle, значения которых указываются в градусах.

Источники света являются объектами Model3D, поэтому можно преобразовывать и анимировать свойства источников света, включая положение, цвет, направление и диапазон.







Windows Forms

WPF

Figure 1
2D Coordinate System

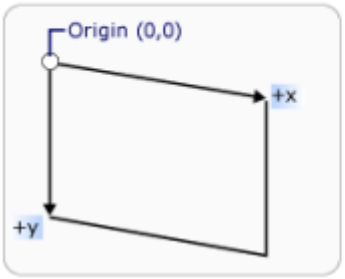
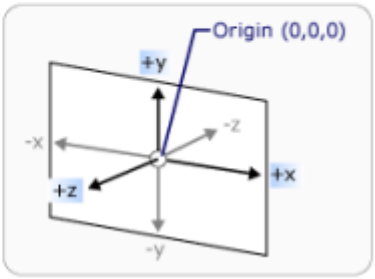
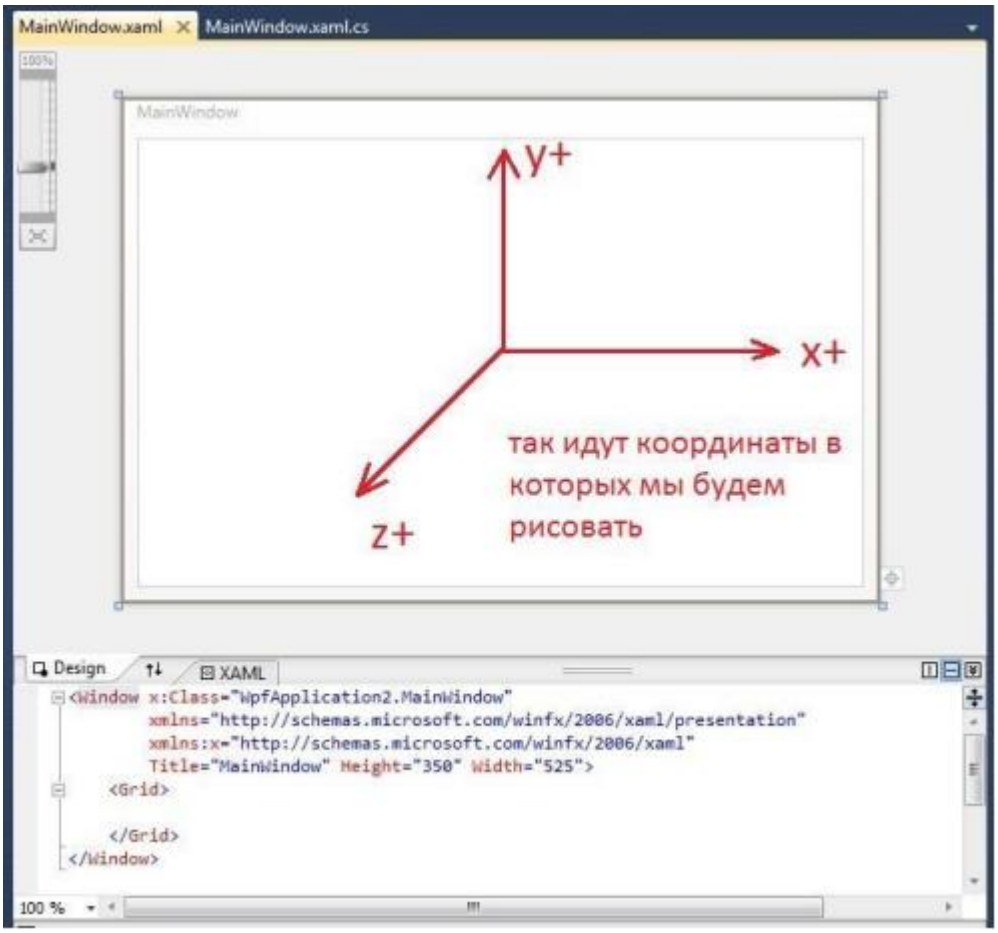


Figure 2
3D Coordinate System

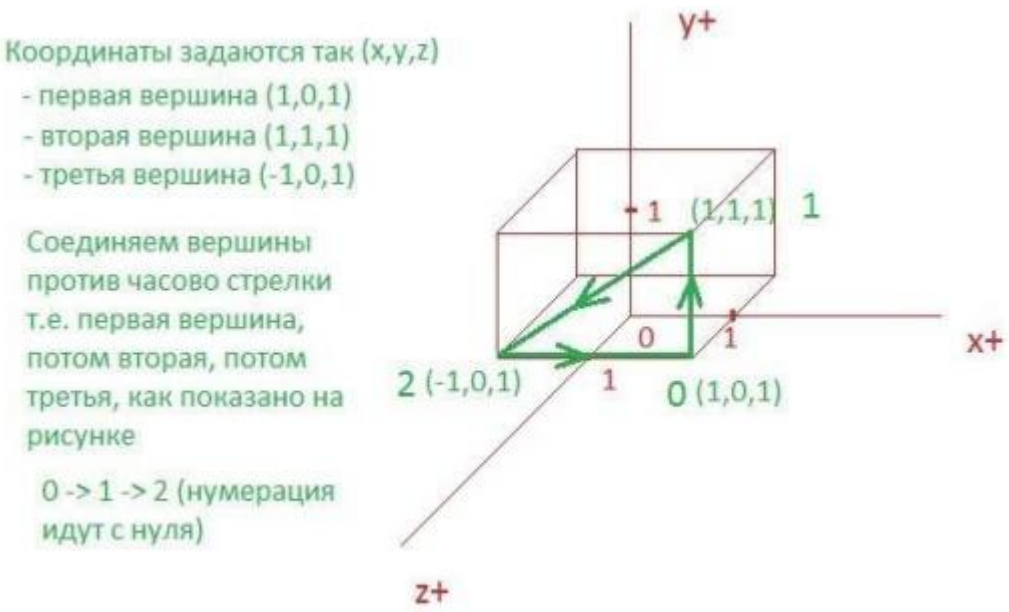


Создание WPF приложения для реализации трехмерных объектов

Необходимо запустить Visual Studio 2017 и создать новый WPF проект. Будет создана форма:



Сначала несколько слов о том, что же мы будем делать: нам необходимо нарисовать куб в пространстве, грани которого состоят из треугольников. Для отрисовки треугольника нам необходимо задать координаты трех точек в трехмерном пространстве, а потом объединить их в треугольник против часовой стрелки. Т.к. WPF работает с векторными величинами, то все то, что мы будем рисовать, помещается в куб со сторонами в 1 (единицу), потому нет необходимости использовать большие величины.



В файл *MainWindow.xaml* добавим следующий код

```
<!--Создаем класс, подключаем необходимые нам библиотеки, задаем заголовок окна, ширину и высоту окна-->
<Window x:Class="Wpf3DTest.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="WPF 3D Test" Height="500" Width="800">
<!--Создаем элемент Grid, задаем фон, и функции обработки нажатий мыши -->
<Grid Background="White" MouseWheel="Grid_MouseWheel" MouseDown="Grid_MouseDown" MouseUp="Grid_MouseUp"
      MouseMove="Grid_MouseMove">
    <Viewport3D x:Name="viewport">
        <!--Создаем элемент Камера, даем ему название, куда он смотрим, откуда смотрит, и попе видимости-->
        <Viewport3D.Camera>
            <PerspectiveCamera x:Name="camera"  LookDirection="0,0,-1" Position="0,0,10" FieldOfView="50" />
        </Viewport3D.Camera>
        <!--Создаем модель, которую будет потом отображать-->
        <ModelVisual3D x:Name="model">
            <ModelVisual3D.Content>
                <Model3DGroup x:Name="group">
                    <!--Задаем цвет света и положение источника света-->
                    <AmbientLight Color="Gray" />
                    <DirectionalLight Color="White" Direction="-5,-5,-10" />
                </Model3DGroup>
            </ModelVisual3D.Content>
        </ModelVisual3D>
    </Viewport3D>
</Grid>
</Window>
```

А в файл *MainWindow.xaml.cs* следующий код:

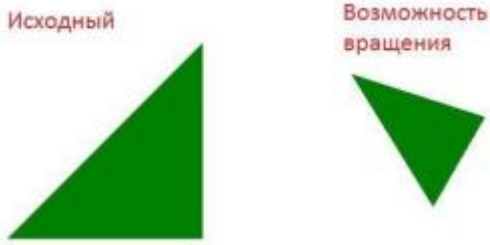
```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Media.Media3D;

namespace Wpf3DTest
{
    public partial class MainWindow : Window
    {
        private GeometryModel3D mGeometry; //геометрическая модель, которую будем рисовать
        private bool mDown; //переменная для проверки нажатия левой клавиши мыши
        private Point mLastPos; //переменная необходимая нам для работы с камерой

        public MainWindow()
        {
            InitializeComponent(); //инициализация программы
            BuildSolid(); //вызов нашей функции которая будет рисовать
        }

        private void BuildSolid()
        {
            MeshGeometry3D mesh = new MeshGeometry3D(); //создаем сетку на основе которой будет создана модель
            //Добавляем вершины сетки
            mesh.Positions.Add(new Point3D(-1, -1, 1)); //0
            mesh.Positions.Add(new Point3D(1, -1, 1)); //1
            mesh.Positions.Add(new Point3D(1, 1, 1)); //2

            //создаем стороны куба из треугольников
            mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(2);

            mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.Green)); //создаем модель из сетки
            mGeometry.Transform = new Transform3DGroup(); //создаем трансформацию для нашей модели
            group.Children.Add(mGeometry); //группируем наши модели (если их будет много, для их общего освещения,
                                                                                    //преобразования и т.д.)
        }
```

Далее

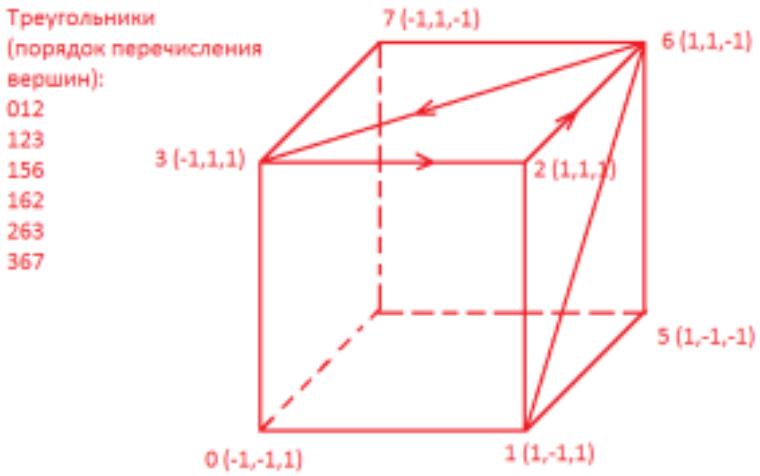
```
//отдаление камеры при прокрутке колесика мыши
private void Grid_MouseWheel(object sender, MouseWheelEventArgs e)
{
    camera.Position = new Point3D(camera.Position.X, camera.Position.Y, camera.Position.Z - e.Delta / 500D);
}
//вращение нашей модели при нажатой левой кнопке мыши и передвижении мыши
private void Grid_MouseMove(object sender, MouseEventArgs e)
{
    if (mDown)
    {
        Point pos = Mouse.GetPosition(viewport);
        Point actualPos = new Point(pos.X - viewport.ActualWidth / 2, viewport.ActualHeight / 2 - pos.Y);
        double dx = actualPos.X - mLastPos.X, dy = actualPos.Y - mLastPos.Y;

        double mouseAngle = 0;
        if (dx != 0 && dy != 0)
        {
            mouseAngle = Math.Asin(Math.Abs(dy) / Math.Sqrt(Math.Pow(dx, 2) + Math.Pow(dy, 2)));
            if (dx < 0 && dy > 0) mouseAngle += Math.PI / 2;
            else if (dx < 0 && dy < 0) mouseAngle += Math.PI;
            else if (dx > 0 && dy < 0) mouseAngle += Math.PI * 1.5;
        }
        else if (dx == 0 && dy != 0) mouseAngle = Math.Sign(dy) > 0 ? Math.PI / 2 : Math.PI * 1.5;
        else if (dx != 0 && dy == 0) mouseAngle = Math.Sign(dx) > 0 ? 0 : Math.PI;

        double axisAngle = mouseAngle + Math.PI / 2;

        Vector3D axis = new Vector3D(Math.Cos(axisAngle) * 4, Math.Sin(axisAngle) * 4, 0);

        double rotation = 0.01 * Math.Sqrt(Math.Pow(dx, 2) + Math.Pow(dy, 2));

        Transform3DGroup group = mGeometry.Transform as Transform3DGroup;
        QuaternionRotation3D r = new QuaternionRotation3D(new Quaternion(axis, rotation * 180 / Math.PI));
        group.Children.Add(new RotateTransform3D(r));
        mLastPos = actualPos;
    }
}
```

```
        //обрабатываем нажатие мыши
        private void Grid_MouseDown(object sender, MouseButtonEventArgs e)
        {
            if (e.LeftButton != MouseButtonState.Pressed) return;
            mDown = true;
            Point pos = Mouse.GetPosition(viewport);
            mLastPos = new Point(pos.X - viewport.ActualWidth / 2, viewport.ActualHeight / 2 - pos.Y);
        }
        //обрабатывает прекращение нажатия мыши
        private void Grid_MouseUp(object sender, MouseButtonEventArgs e)
        {
            mDown = false;
        }
    }
}
```

Запускаем программу и видим следующий результат:



Далее нарисуем из треугольников куб:



для этого изменим функцию BuildSolid() следующим образом:

```
private void BuildSolid()
{
    MeshGeometry3D mesh = new MeshGeometry3D(); //создаем сетку для создания модели
    //Добавляем вершины сетки
    mesh.Positions.Add(new Point3D(-1, -1, 1)); //0
    mesh.Positions.Add(new Point3D(1, -1, 1)); //1
    mesh.Positions.Add(new Point3D(1, 1, 1)); //2
    mesh.Positions.Add(new Point3D(-1, 1, 1)); //3
    mesh.Positions.Add(new Point3D(-1, -1, -1)); //4
    mesh.Positions.Add(new Point3D(1, -1, -1)); //5
    mesh.Positions.Add(new Point3D(1, 1, -1)); //6
    mesh.Positions.Add(new Point3D(-1, 1, -1)); //7

    //создаем стороны куба из треугольников
    mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(2); //спереди
    mesh.TriangleIndices.Add(2); mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(5); mesh.TriangleIndices.Add(6); //справа
    mesh.TriangleIndices.Add(1); mesh.TriangleIndices.Add(6); mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(2); mesh.TriangleIndices.Add(6); //сверху
    mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(6); mesh.TriangleIndices.Add(7);
    mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(5); mesh.TriangleIndices.Add(1); //снизу
    mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(4); mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(3); mesh.TriangleIndices.Add(7); //слева
    mesh.TriangleIndices.Add(0); mesh.TriangleIndices.Add(7); mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(7); mesh.TriangleIndices.Add(6); mesh.TriangleIndices.Add(5); //сзади
    mesh.TriangleIndices.Add(7); mesh.TriangleIndices.Add(5); mesh.TriangleIndices.Add(4);

    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.Green)); //создаем модель из сетки
    mGeometry.Transform = new Transform3DGroup(); //создаем трансформацию для нашей модели
    group.Children.Add(mGeometry); //группируем наши модели (если их будет много, для их общего освещения,
                                                                    //преобразования и т.д.)
}
```

В результате получим: