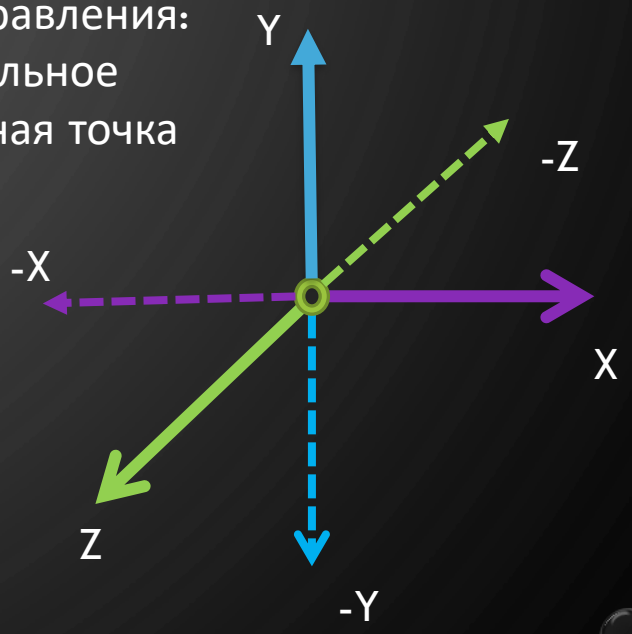
3D-ГРАФИКА

QT 3D

1. ОСНОВЫ

Набор модулей Qt 3D предоставляет возможность моделирования трехмерных сцен в режиме реального времени. Все эти модули можно использовать как из языка C++, так и из языка QML.

Трехмерное пространство на дисплее - это виртуальная модель нашей среды обитания. Для определения местоположения в нем необходимы три направления: X, У и Z. Первое задает ширину, второе- высоту, а третье- глубину. Виртуальное пространство определено *сценой*, в центре которой расположена начальная точка отсчета по этим направлениям.



За реализацию виртуальной сцены (показ трехмерной сцены с объектами в Qt 3D) отвечает элемент Scene3D из модуля QtQuick.Scene3D.

Все, что находится внутри сцены: *3D-объекты*, *камера*, *свет* - все это должно быть выражено в форме *компонентов сущностей*.

2. ОСНОВЫ

Система компонентов сущностей (*Entity Component System, ECS*) - это шаблон, который часто используется в игровых движках, симуляторах и был заимствован в Qt 3D. Её основное преимущество - в гибкости изменения поведения сущностей, которое может проводиться в режиме реального времени, путем добавления либо удаления компонентов без прерывания процесса работы программы.

Сущности реализуются в QML элементами *Entity*. Сама по себе сущность не несет в себе ни поведения, ни характеристик. *Поведение* может быть добавлено к сущности путем объединения одного или нескольких компонентов. Сущности – это контейнеры, в которые можно добавлять компоненты. Например, можно добавить геометрический образ объекта и к нему добавить элемент поведения - *трансформации*.

Для того чтобы отобразить в трехмерной сцене элемент, его необходимо оформить в виде *сущности*. Т.е. для отображения 3D-объекта понадобится объединить в элементе сущности его геометрический *образ* (элемент *Mesh*), *материал* (элемент *Material*), дающий оболочку, и *трансформации* (элемент *Transform*).

```
Entity {
    Mesh { id: myMesh; ... }
    PhongMaterial { id: myMaterial ... }
    Transform { id: myTansform; ... }
    component: [myMesh, myMaterial,
                myTransform]
}
```

3. СВЕТ

Глубина тени 3D-объектов очень зависит от источников света. Свет играет в процессе визуализации трехмерных сцен важную роль. Именно он придает 3D-объектам реалистичность и создает настроение восприятия всей трехмерной сцены.

*Чрезмерно сильное освещение может сделать сцену плоской. А использование света в виде луча прожектора может привлечь внимание к определенному месту сцены или объекта. Интенсивный цвет способен сформировать острые и четкие тени от объектов, а мягкий свет - более размытые, что подчеркнет пространство и создаст еще большую реалистичность восприятия.

Qt 3D предоставляет три вида источников света:

- *точечный свет* - реализован элементом *PointLight*, излучает свет от центра и сразу во всех направлениях. Более далеко расположенные от источника объекты освещены меньше всего. Этот тип света отлично подходит для имитации освещения от лампочки, свечи или факела;
- *направленный свет* - реализован элементом *DirectionalLight*, излучает множество лучей, которые поступают извне и освещают сцену с бесконечного расстояния. Все объекты сцены получают одинаковые порции света вне зависимости от их расположения. Этот тип света подходит для имитации света от Солнца и Луны;
- *прожекторный свет* - реализован элементом *SpotLight*, излучает свет из центральной точки, лучи расходятся в виде конуса. Этот тип света подходит для имитации настольных ламп, прожекторов, фар автомобилей, студийного освещения и фонарей.

4. СВЕТ

Все источники света имеют общие свойства:

- *цвет* (свойство color) - управление цветом освещения. Для большей реалистичности не рекомендуется использовать однотонные цвета. Например, не стоит задавать в качестве дневного цвета чисто белый цвет. Лучше, если добавить к нему какой-нибудь оттенок, например , желтый;

- *интенсивность* (свойство intensity) - управляет яркостью освещения. Для достижения хороших результатов рекомендуется начинать с минимального значения интенсивности и понемногу ее увеличивать, пока не будет достигнут оптимальный результат.

5. КАМЕРА

Камеры в виртуальном пространстве, нужны, чтобы получить проекцию реального (виртуального) трехмерного мира на плоскости.

Камеры в Qt 3D, в отличие от камер реального мира, это не визуальные объекты, то есть их не видно на сцене. Их можно размещать на любой позиции трехмерной сцены и регулировать угол захвата для получения необходимого изображения. Положение камеры, как и любого другого объекта сцены, можно изменять во времени, то есть анимировать. Эту возможность можно использовать, например, для того чтобы показать движение внутри или вдоль объекта (здания, туннеля и т.д)

тип для проекции:

- PerspectiveProjection - перспективная
- OrthographicProjection – ортогональная
- FrustumProjection - пирамидальная

```
Camera {
    projectionType: CameraLens.PerspectiveProjection // тип проекции
    fieldOfView: 90                                  // угол захвата
    position: Qt.vector3d( 0.0, 0.0, 40.0 )
}
```

6. 3-D ОБЪЕКТЫ

3D-объекты состоят из многоугольников, которые задаются координатами вершин, описания поверхностей и нормальных векторов к ним.

Для загрузки 3D-объектов Qt 3D использует формат OBJ. Он может импортироваться и экспортироваться практически во все популярные программы создания 3D-графики - такие как, например, Maya, 3ds Max и Blender.

Загружать данные из OBJ-файлов позволяет элемент ***Mesh***. Вот как можно загрузить файл “pyramid.obj”:

```
Mesh {
    id: mesh
    source: "pyramid.obj“
}
```

7. 3D-ОБЪЕКТЫ

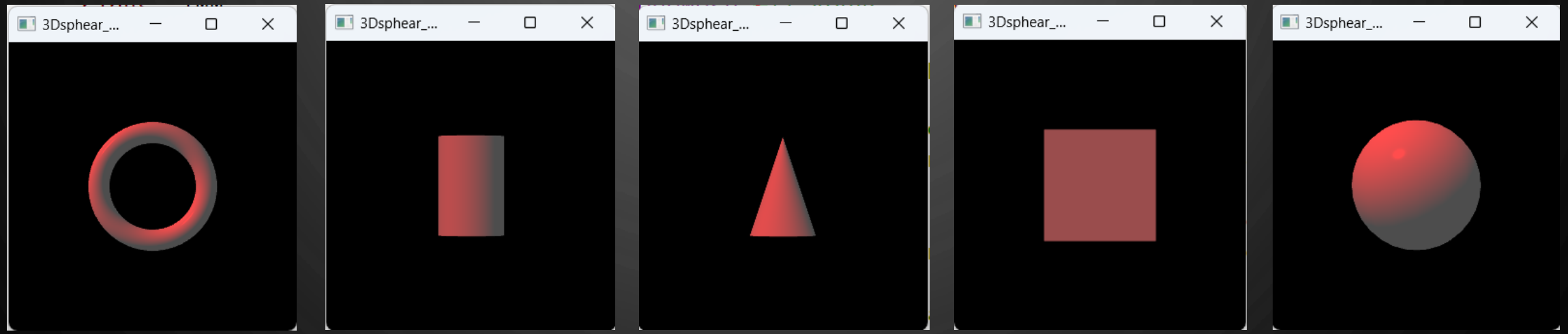
Модуль *Qt3D.Extras* предоставляет коллекцию стандартных форм трехмерных объектов. К этим формам относятся наиболее распространенные объекты: сфера, параллелепипед, цилиндр и др

```
SphereMesh {   // сфера
    radius: 3    //радиус
}// параллелепипед
CuboidMesh { //размеры лицевых сторон
      yzMeshResolution: Qt.size(2, 2)
      xzMeshResolution: Qt.size(2, 2)
      xyMeshResolution: Qt.size(2, 2)
}
PlaneMesh {// плоскость
       width: 50
       height: 50
}
TorusMesh {// тор
      radius: 5 //внешний радиус
      minorRadius: 1 // внутренний радиус
      rings: 100       //количество колец корпуса
}
```

```
ConeMesh {// конус
      topRadius: 0 //радиус верхнего основания
      bottomRadius: 1 //радиус нижнего основания
      length: 3 //высота
      rings: 50 //количество колец корпуса
}

CylinderMesh { // цилиндр
      radius: 1 //радиус оснований
      length: 3 //высота
      rings: 100 //количество колец корпуса
}
```

8. 3D-ОБЪЕКТЫ



9. МАТЕРИАЛЫ

Для отображения геометрического образа нужно добавить в сущность **материал** и тем самым создать для трехмерного объекта оболочку. Материалы отличаются друг от друга способностью отражать свет.

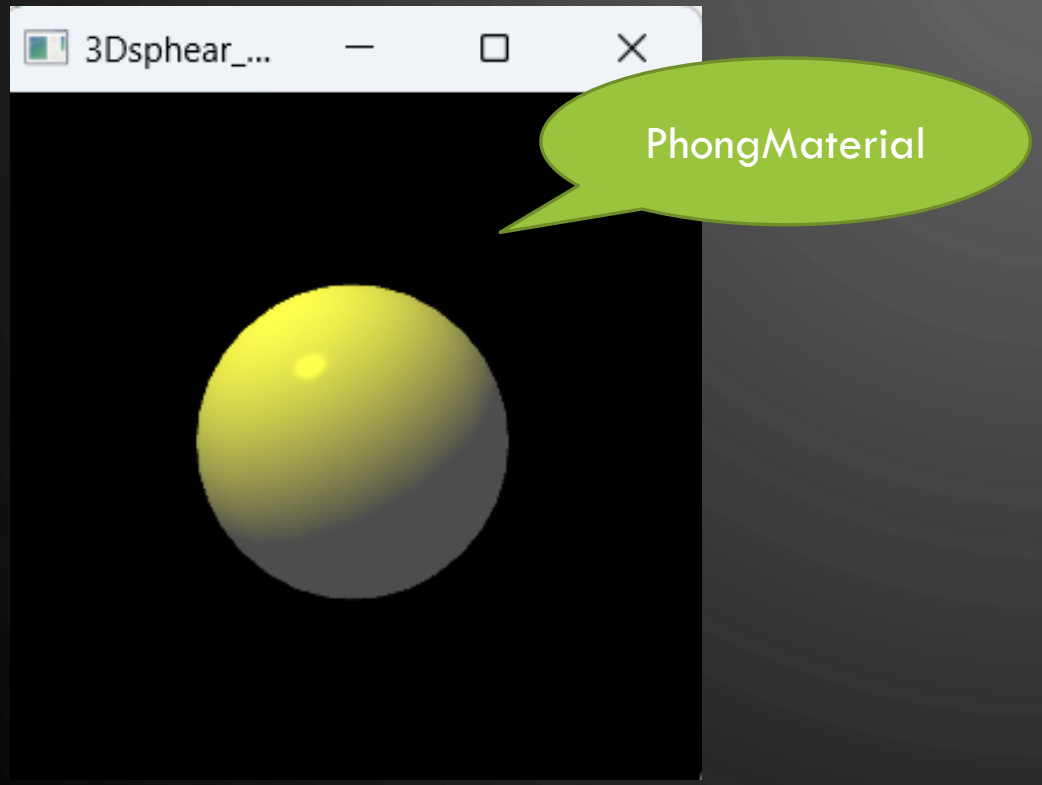
Qt 3D представляет следующие элементы для материалов:

- элемент PhongMaterial - представляет гладкие поверхности. Это довольно яркий материал, обеспечивающий на поверхности объектов формирование бликов, которые придают им блестящий или глянцевый вид. В качестве эквивалента этому материалу хорошо подойдут пластмасса и металл;
- элемент GoochMaterial - представляет поверхность без сглаживания, цвет которой остается постоянным. Отображение этого материала происходит быстрее, чем в случае с *PhongMaterial*, поэтому его замечательно можно использовать для тестовых визуализаций. В качестве эквивалента этому материалу хорошо подойдут картон, бумага и некоторые типы древесины.

Эти два материала имеют следующие общие свойства:

- Св-во diffuse (*Диффузия*) - задает основной цвет поверхности объекта значением типа *color*;
- Св-во shininess (*Свечение*) - задает уровень свечения поверхности от 0 до 1;
- Св-во alpha (*Прозрачность*) - управляет способностью поверхности объекта пропускать опред. кол-во света

10. МАТЕРИАЛЫ



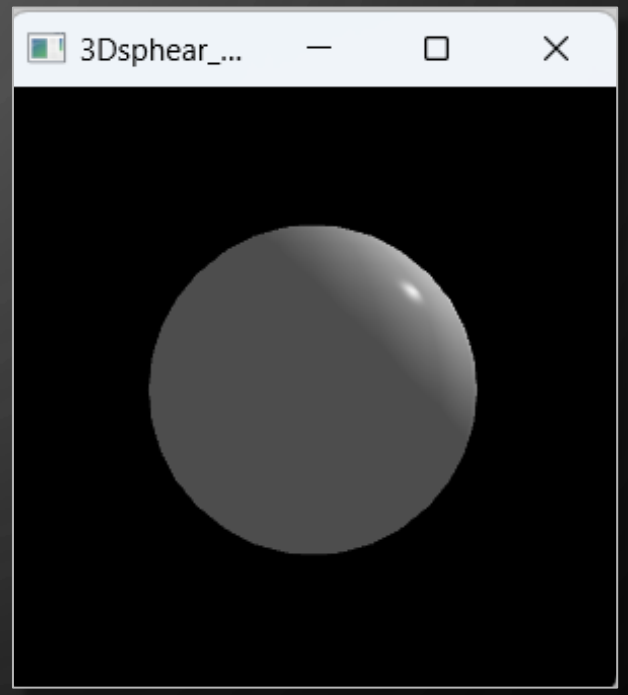
11. ОТОБРАЖЕНИЕ 3D-СФЕРЫ

```
QT += quick qml 3dcore 3drender 3dinput 3dextras 3dquick 3dquickextras
```

main.cpp

```
#include <QGuiApplication>
#include <QQuickView>

int main(int argc, char** argv) {
    QGuiApplication app(argc,argv);
    QQuickView view;
    view.resize(300, 300);
    view.setResizeMode(QQuickView::SizeRootObjectToView);
    view.setSource (QUrl ( "qrc:/main.qml" ));
    view.show();
    return app.exec();
}
```



12. ОТОБРАЖЕНИЕ 3D-СФЕРЫ

main.qml

```
import QtQuick 2.15
import Qt3D.Core 2.15
import Qt3D.Render 2.15
import Qt3D.Input 2.15
import Qt3D.Extras 2.15
import QtQuick.Scene3D 2.0
```

```
Rectangle {
    color: “black”
    Scene3D {
        anchors.fill: parent ;   focus: true
        aspects: ["input", "logic" ]
        cameraAspectRatioMode: Scene3D.AutomaticAspectRatio
        Entity {
            Camera { //Камера
              id: camera ;   nearPlane : 0.1;  farPlane : 1000.0; fieldOfView: 10
              position: Qt.vector3d( 0.0, 0.0, 50.0 )
            }
            FirstPersonCameraController { //Управление положением камеры
                  camera: camera
                  linearSpeed: 1000.0 ; acceleration: 0.1; deceleration: 1.0
            }
            components: [ RenderSettings { activeFrameGraph: ForwardRenderer {
                                                                camera: camera;
                                                            clearColor: "transparent" }
                                    }, InputSettings { }, DirectionalLight { color: "#afafff“ }
                                ]
...
} } }
```

```
Entity { //Сущность сферы
    PhongMaterial {
        id: phongMaterial
        ambient: Qt.rgba( 0.3, 0.3, 0.3, 1.0 )
        diffuse: Qt.rgba( 1, 1, 1, 1 )
    }
    SphereMesh {
        id: sphereMesh
        radius: 6
    }
    components: [sphereMesh, phongMaterial]
}
```

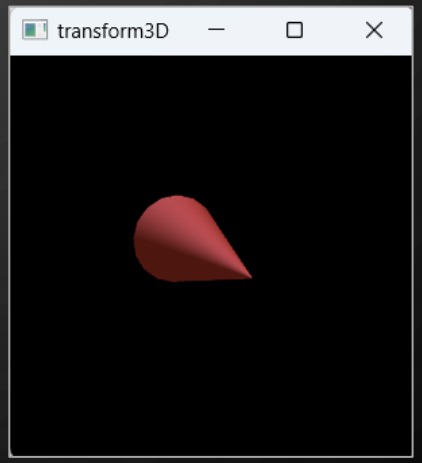
13. ТРАНСФОРМАЦИЯ

```
Entity {
    ConeMesh { //Геометрия объекта
        id: coneMesh1
        topRadius: 0
        bottomRadius: 1
        length: 5
        rings: 50
    }
    Transform { //Изменение места положения
        id: coneTransform1
        scale: 0.9
        translation: Qt.vector3d(0, 14, 4)
        rotationX: 100;     rotationY: 20
    }
    PhongMaterial {//Материал
        id: coneMaterial1
        shininess: 0.9
        ambient: Qt.rgba(0.3, 0.3, 0.3, 1.0) // отражение
        diffuse: Qt.rgba( 1.0, 0.3, 0.2, 1 )    }
    components: [coneMesh1, coneMaterial1, coneTransform1]
}
```

Фрагмент программы «Конус с трансформацией"

Трансформация - это изменения положения объекта вдоль любых из трех осей или плоскостей, а также его повороты вокруг собственной оси и изменение его размеров. За все эти операции отвечает элемент Transform, осуществляющий их с помощью свойств: translation, rotation и scale.

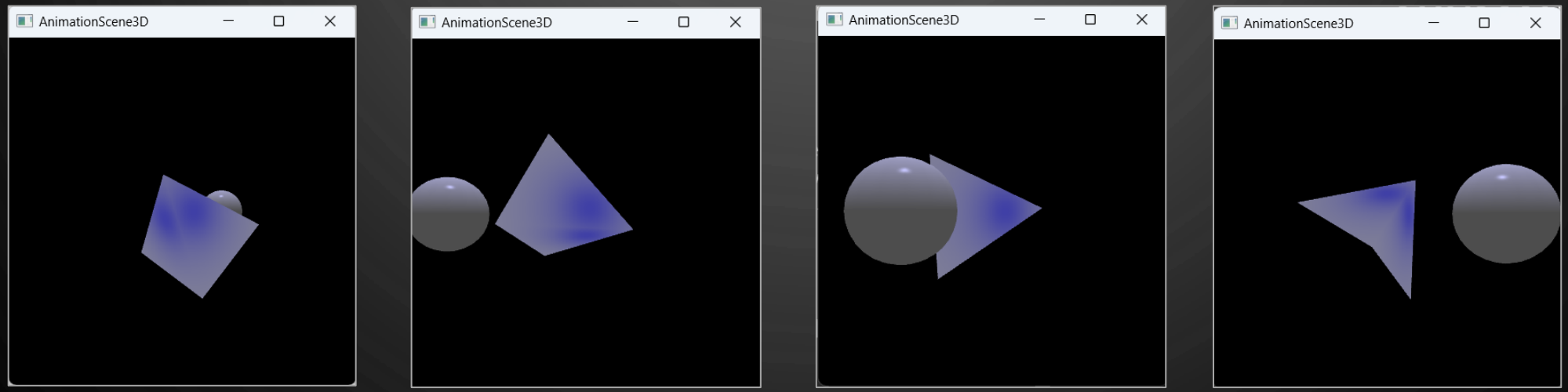
*Трансформации изменения местоположения и угла поворота можно также применять к элементам *света и камерам*.



14. АНИМАЦИЯ

Для анимации трехмерных объектов можно задействовать уже знакомые нам приёмы, которые были использованы для анимации элементов на плоскости .

Воспользуемся знакомым элементом *NumberAnimation* и применим его к трехмерным объектам сферы и пирамиды.



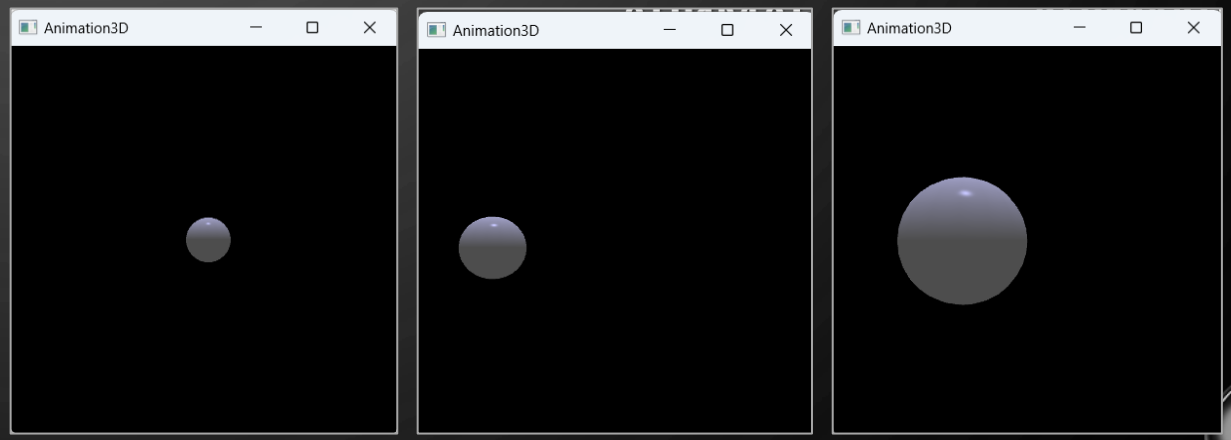
15. АНИМАЦИЯ

Анимация сферы (вращение вокруг оси Y)

```
Entity {
    PhongMaterial { //Материал
        id: phongMaterial
        ambient: Qt.rgba ( 0.3, 0.3, 0.3, 1.0 )
        diffuse: Qt.rgba ( 1, 1, 1, 1)
    }
    SphereMesh { //Геометрия объекта сферы
        id: sphereMesh
        radius: 6
    }
    Transform { //Изменения положения объекта сферы
        id: sphereTransform
        property real myParam: 0 // угол поворота объекта
        matrix: {
            var mat = Qt.matrix4x4();
            mat.rotate ( myParam, Qt.vector3d(0, 1, 0))
            mat.translate ( Qt.vector3d(24, 0, 0));
            return mat;
        }
    }
components: [sphereMesh, phongMaterial, sphereTransform]
. . .
}
```

Ось вращения

```
NumberAnimation { //Анимация объекта сферы
    target: sphereTransform
    property: "myParam"
    duration: 10000
    from: 0
    to: 360
    loops: Animation.Infinite
    running: true
}
```

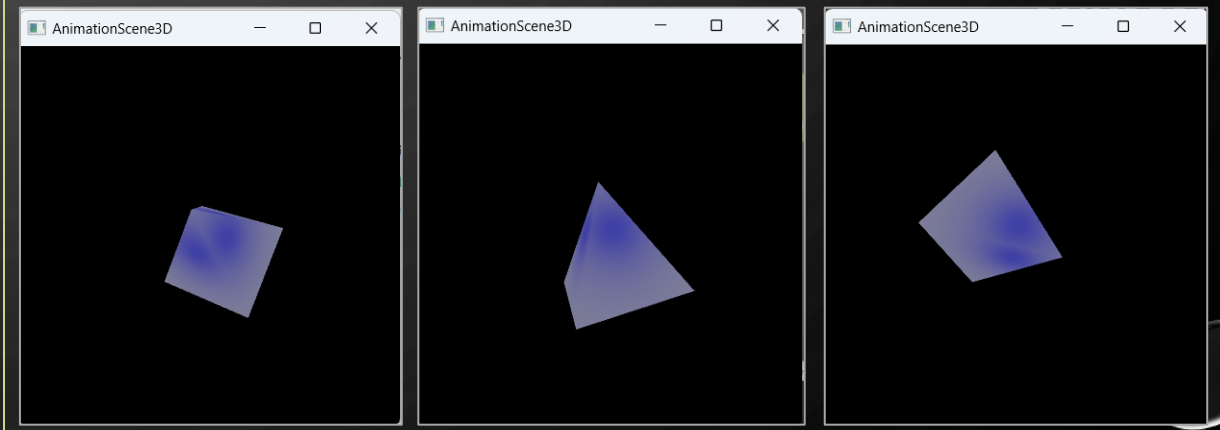


16. АНИМАЦИЯ

Анимация пирамиды
(вращение вокруг трех осей)

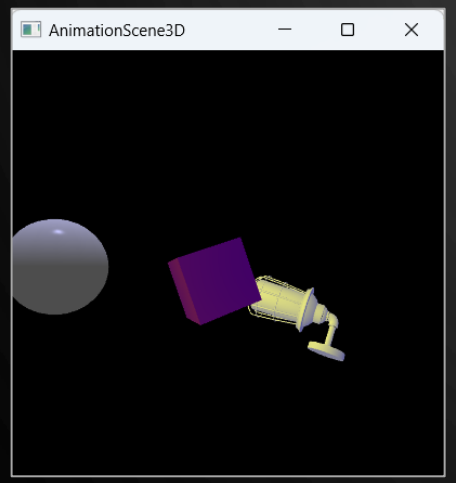
```
Entity {
    GoochMaterial {//Материал
       id: goochMaterial
      diffuse: Qt.rgba( 1, 1, 1, 1 )
    }
    Mesh { //Геометрия объекта пирамиды
      id: pyrarnidMesh
      source: "qrc: /pyrarnid.obj"
    }
    Transform { // Повороты объекта пирамиды
        id: pyrarnidTransform
        property real myRotation: 0
        matrix: {
            var mat = Qt.matrix4x4();
            mat.rotate(myRotation, Qt.vector3d(1, 1, 1))
            mat.scale(Qt.vector3d(10, 10, 10) );
            return mat;
    }
}
components: [pyrarnidMesh, goochMaterial, pyrarnidTransform]
. . .
}
```

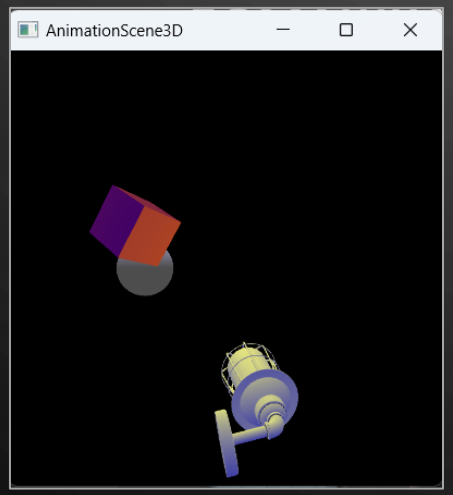
```
//Анимация объекта пирамиды
NumberAnimation {
    target: pyrarnidTransform
    property: "myRotation"
    duration: 10000
    from: 0
    to: 360
    loops: Animation.Infinite
    running: true
}
```



17. ДОМАШКА #14

Создание сложной 3D-сцены с анимацией. В сцене должны использоваться сложные объекты, созданные из элементарных 3D-фигур или объекты в формате .obj





МОЖЕТ ПРИГОДИТЬСЯ

- HTTPS://WWW.FREEPIK.COM/3D-MODELS