OpenGL

Часть 1. Введение

Что такое OpenGL

- OpenGL(Open Graphics Library) – это независящий от языка программирования и платформы программный интерфейс для написание приложений, использующих 2D и 3D графику
- Написанием спецификаций занимается Khronos Group
- Используется для написания игр(Half-Life 1,2,3 o_0, Bioshock, Counter-Strike, …), специализированного ПО(Photoshop, Google Earth, Blender, …) , визуализации в научных исследованиях и т. д.

Реализации OpenGL

- На данный момент существуют реализации практически для всех платформ
- Для мобильных платформ(iOs, Android) существует собственная спецификация OpenGL ES, учитывающая их аппаратные особенности
- Большинство современных браузеров поддерживают WebGL - аналог OpenGL ES
- Готовится спецификация Vulkan API для объединения OpenGL и OpenGL ES

Зачем нам OpenGL

- OpenGL позволяет не обращать внимания на особенности аппаратуры и драйверов, предоставляя единый API для разработчика
- Если какая-то возможность не поддерживается аппаратурой/драйвером то OpenGL эмулирует её
- Реализации OpenGL проходят специальные тесты на совместимость и эффективность использования оборудования

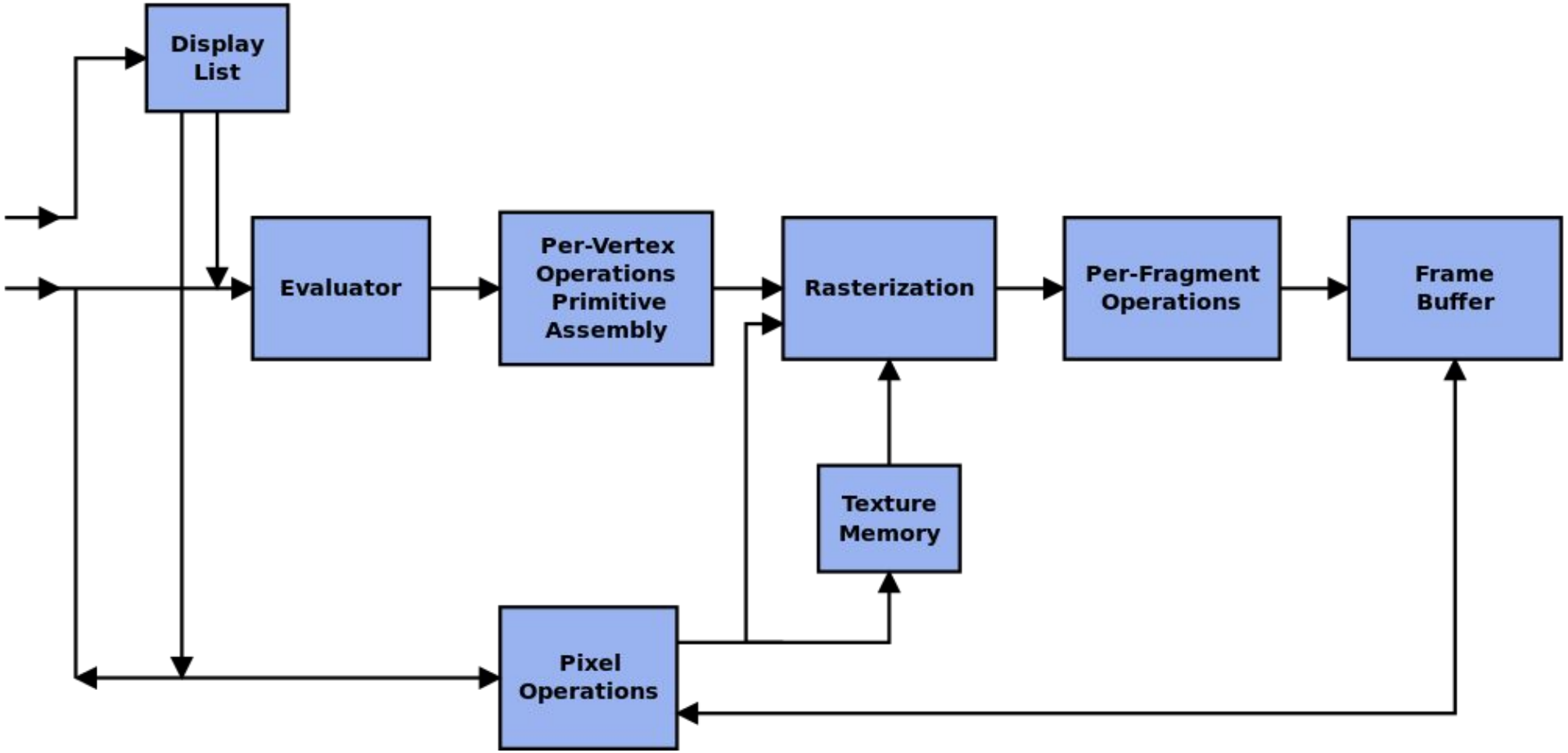
Архитектура OpenGL

- Основной принцип работы: получение набора векторных графических примитивов(с математической обработкой) – точек, линий и треугольников для построения растровой картинки на экране и/или в памяти
- Векторные трансформации и растеризация выполняются графическим конвейером (graphics pipeline), который по сути представляет собой дискретный автомат
- Почти все команды OpenGL делятся на 2 группы:
  1. Добавление примитивов на вход в конвейер
  2. Конфигурация конвейера для выполнения трансформаций
- Императивный подход, т.е. Описание точной последовательности шагов для получения конечной растровой картинки

Основные положения

- Растеризация - это перевод изображения, описанного векторным форматом в пиксели или точки, для вывода на дисплей
- Render – полный процесс отображения растровой картинки на экране
- Render Frame(или просто Frame) – часть процесса рендера, отображающая картинку в конкретном состоянии в данный момент времени
- Texture – изображение, загружаемое в память для использования в процессе рендера
- Framebuffer – набор буфферов в памяти устройства, содержащий всю информацию о данном состоянии на экране
- Render context - поток в котором происходит процессе рендера

Графический конвейер OpenGL



OpenGL в QT

- За работу с OpenGL в QT отвечает модуль QT OpenGL
- Основные классы для работы с OpenGL:
  1. QGLWidget – виджет поддерживающий работу с OpenGL.
  2. QGLFunctions – класс, содержащий все необходимые функции для работы с OpenGL
  3. QGLShaderProgram – класс для работы с шейдерами
  4. QGLTexture - класс для работы с текстурами
  5. QVectror4D/3D/2D,QMatrix(4x4,3x3,2x2), QQuaternion и т.д. – классы для работы с математическими примитивами

QGLWidget

- Виртуальная функция *initializeGL()* - функция для начальной инициализации OpenGL. Вызывается до начала процесса рендера
- Виртуальная функция *resizeGL()* – функция, вызываемая при изменение размера виджета
- Виртуальная функция *paintGL()* – функция, в которой содержится последовательность действий для отображение фрейма

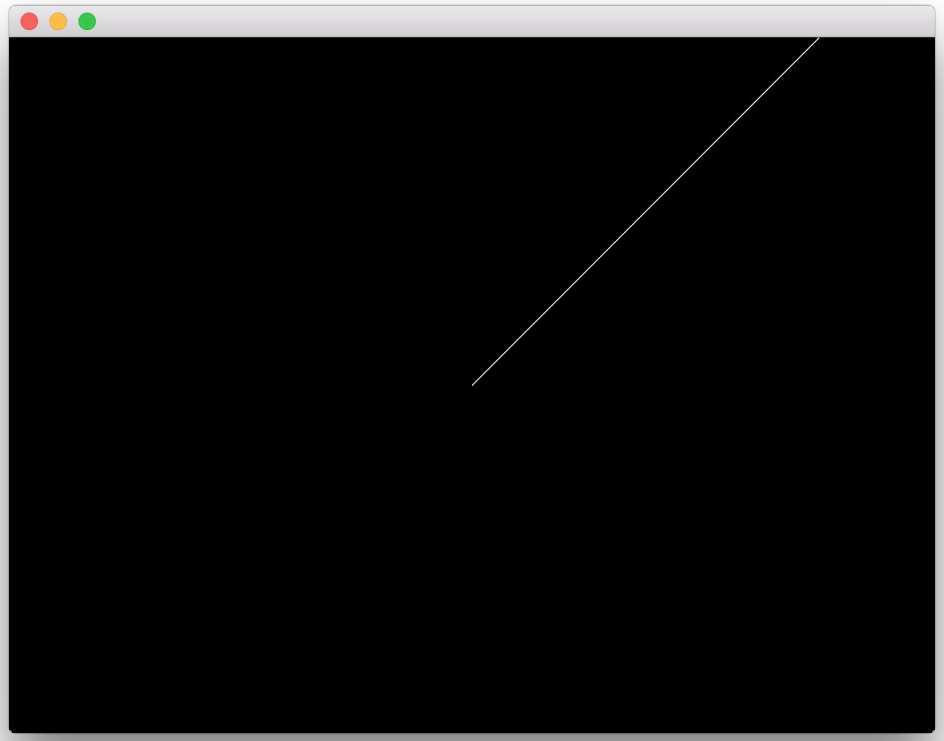
QGLFunctions - основные функции

- glEnable/glDisable(feature) – включает/выключает поддержку определенного свойства контекста OpenGL
- glClear(clear_bitfield) – очищает указанные буфферы
- glClearColor(r,g,b,a) – задает цвет в формате RGBA, который будет использоваться для в момент очистки буффера цвета
- glVertex(2/3/4)(s/f/d/i)[v](…) - указывает координаты вершины в различных форматах
- glBegin(mode) – указывает как использовать данные о вершинах, которые будут заданы после вызова данной функции. После задания всех нужных вершин вызывается glEnd().
- glColor3f/4f[v](…) – указывает какого цвета будут вершины после вызова данной функции

QGLFunctions - основные функции

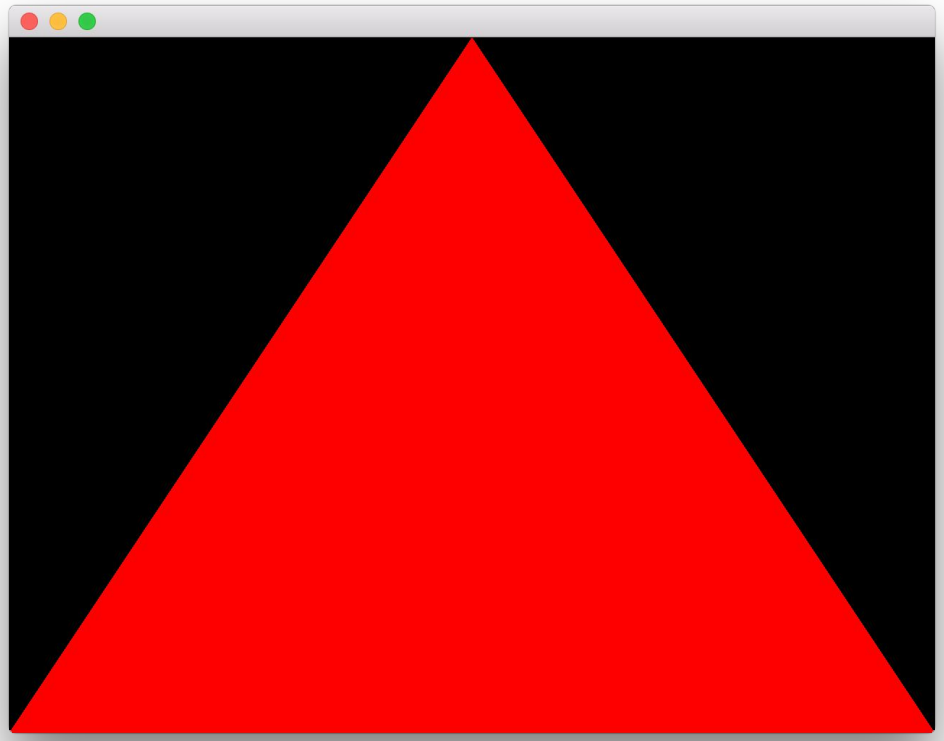
- glMatrixMode(mode) - указывает, в какой стек положить матрицу преобразования, загруженную после вызова этой функции
- glLoadMatrixf – устанавливает матрицу в вершине стека(стек выбран с помощью glMatrixMode) матрицу
- glPush(Pop)Matrix – добавляет(удаляет) матрицу в(из) выбранный стек матриц
- glViewport(x,y,w,h) – указывает положение и размер окна для отрисовки контекста OpenGL

Рисуем линию



```
void MainWindow::paintGL()
{
    // Очищает буфет цвета
    glClear(GL_COLOR_BUFFER_BIT);
    //Задаем толщину линий
    glLineWidth(2.5);
    //Рисуем в режиме линий
    glBegin(GL_LINES);
    //Задаем цвет вершин
    glColor3f(1, 1, 1);
    //Добавляем вершины
    glVertex3f(0, 0, 0);
    glVertex3f(1, 1, 1);
    //Оконцанице рисования в режиме линий
    glEnd();
}
```

Рисуем треугольник

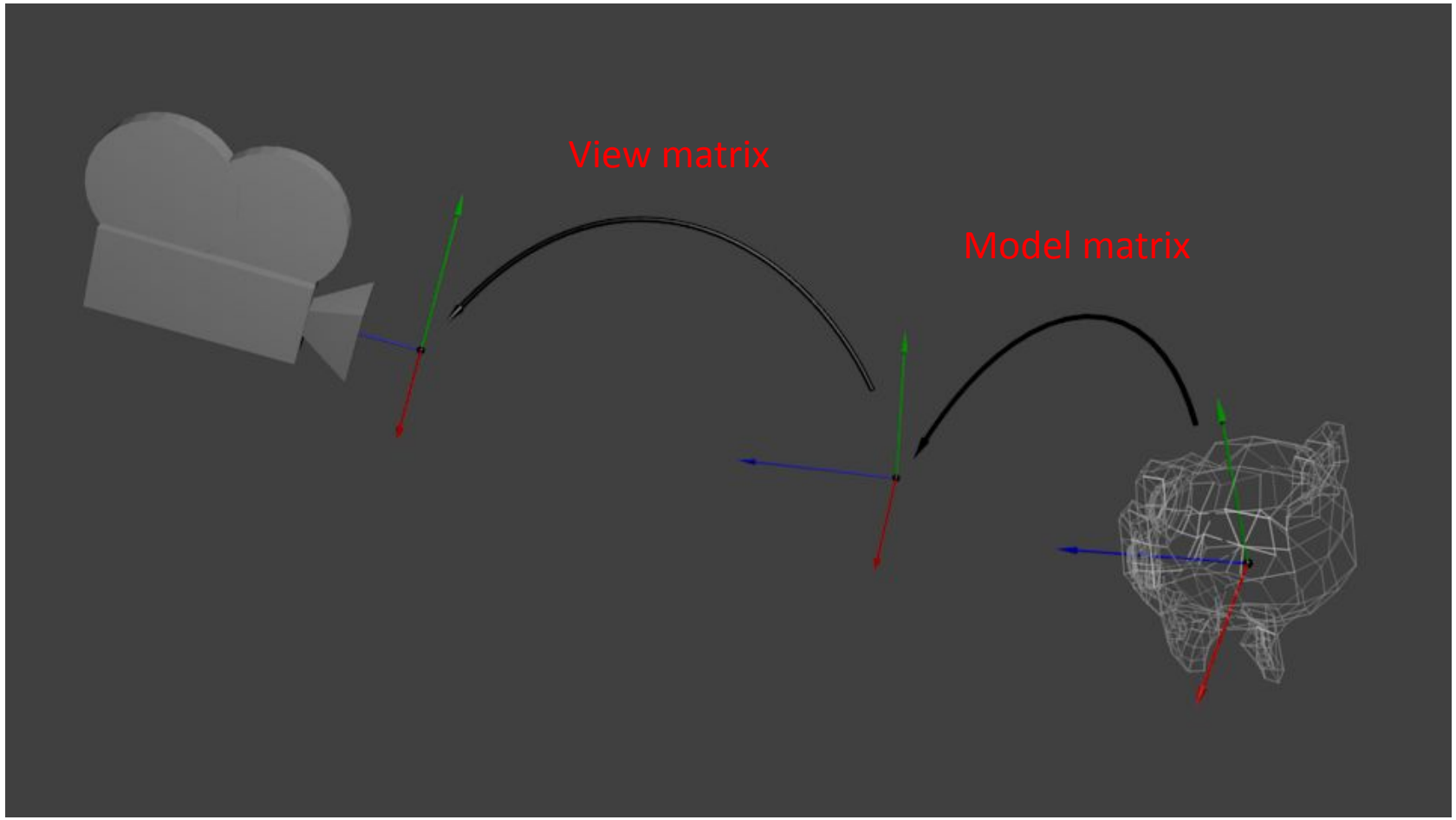


```
void MainWindow::paintGL()
{
    // Очищает буфет цвета
    glClear(GL_COLOR_BUFFER_BIT);
    //Рисуем в режиме треугольников
    glBegin(GL_TRIANGLES);
    //Задаем цвет вершин
    glColor3f(1, 0, 0);
    //Добавляем вершины
    glVertex3f(-1, -1, 0);
    glVertex3f(1, -1, 0);
    glVertex3f(0, 1, 0);
    //Окончанице рисования в режиме треугольников
    glEnd();
}
```

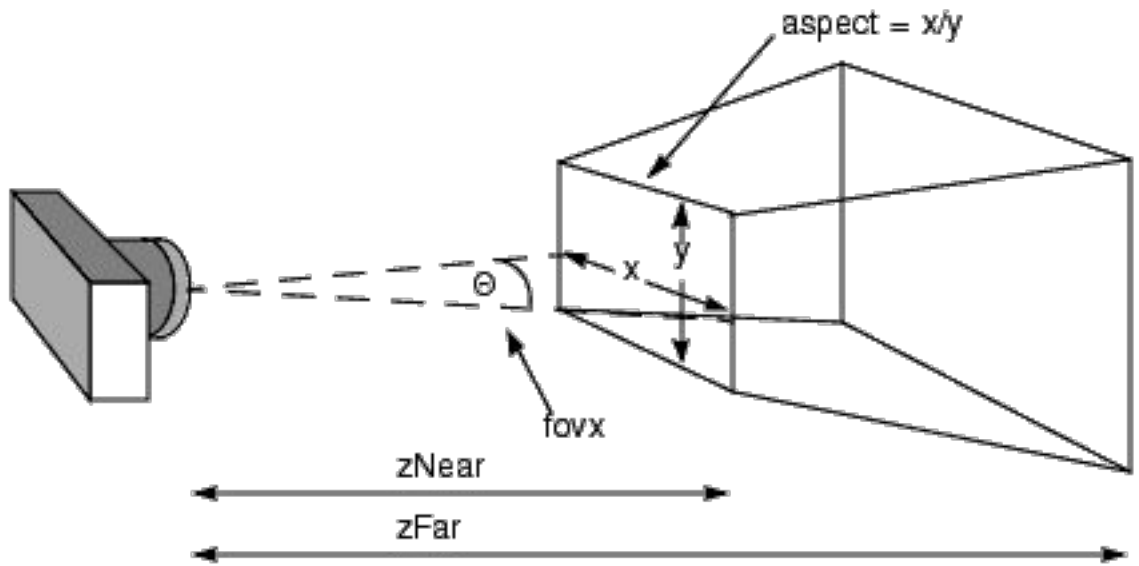
Немного о матрицах в 3D графике

- Матрицы преобразования делятся на 3 типа: Model, View, Projection
- Model matrix – матрица преобразования координат из локальных в глобальные(World coordinates)
- View matrix – матрица преобразования координат из глобальных в координаты взгляда(Camera)
- Projection matrix – матрица преобразования координат из координат камеры в перспективные координаты(составляется по 4 параметрам: Горизонтального поле зрения – в градусах, соотношение сторон окна, расстояние до ближней и дальней плоскости перспективы(Ось z))
- Конечная координата = Projection * View * Model * Vertex

Немного о матрицах в 3D графике

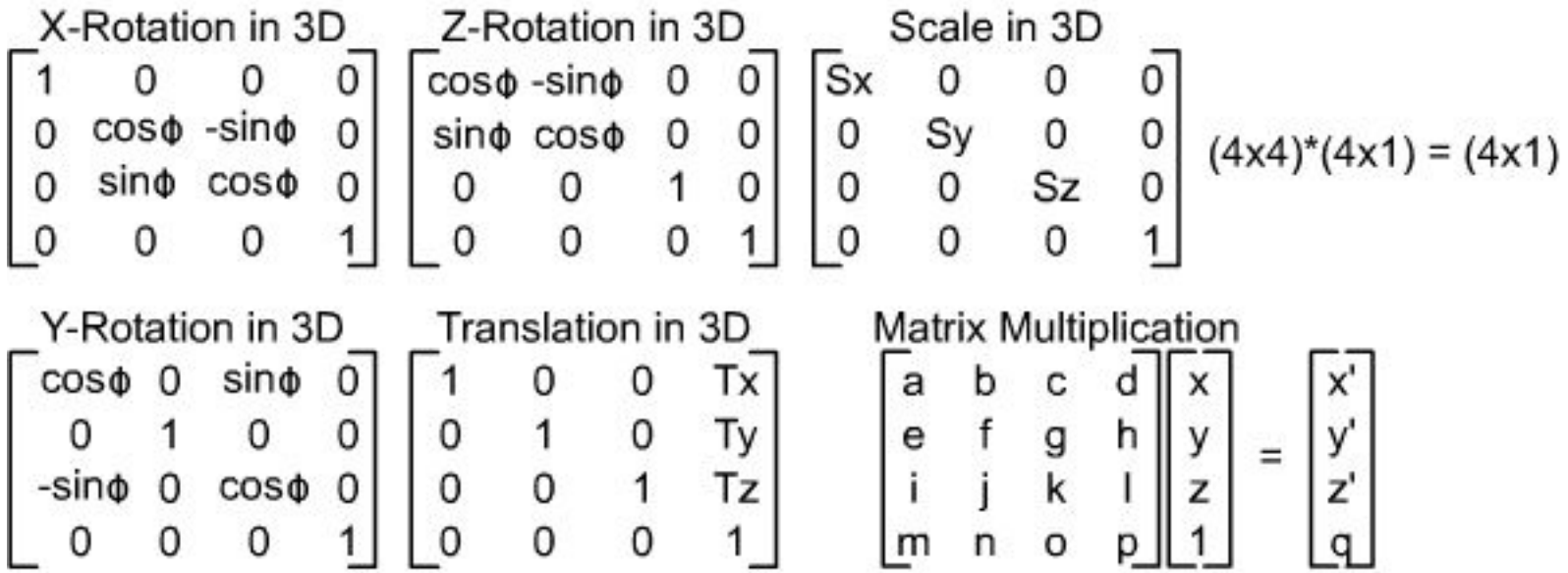


Немного о матрицах в 3D графике



Немного о матрицах в 3D графике

- Наиболее используемый размер матриц – 4x4
- Почему 4x4 ?



- Подматрица 3x3 называется нормальной матрицей

Немного о матрицах в 3D графике

- Часто удобнее вместо матриц поворота использовать кватернионы. Они занимают меньше памяти, а так же меньше затрат на операции с ними.
- Кватернион - это число вида $w + xi + yj + zk$, где
  - $i^2 = j^2 = k^2 = -1$
  - $ij = -ji = k$
  - $jk = -kj = i$
  - $ki = -ik = j$
- Мнимая часть (x, y, z) - вектор $\mathbb{R}^3$ w $\in \mathbb{R}$

Свойства кватернионов

- Ассоциативность
- Не коммутативны
- Определены операции векторного и скалярного умножения, причем $v \times w = \mathrm{vw} + \mathrm{v} \cdot w$
- Если рассмотреть евклидовы координаты (w, x, y, z) , где $w^2 + x^2 + y^2 + z^2 = 1$, то точка (w, x, y, z) представляет собой вращение вокруг оси (x, y, z) на угол $\alpha = 2\arccos(\mathrm{w}) = 2\arcsin\sqrt{x^2 + y^2 + z^2}$, $q = w + xi + yj + zk = \mathrm{w} + (x, y, z) = \cos {}^{\propto}/_{2} + u\, sin {}^{\propto}/_{2}$, где u - единичный вектор
- Тогда qvq^{-1} - вращает вектор v на угол $\propto$ вокруг оси u

Свойства кватернионов

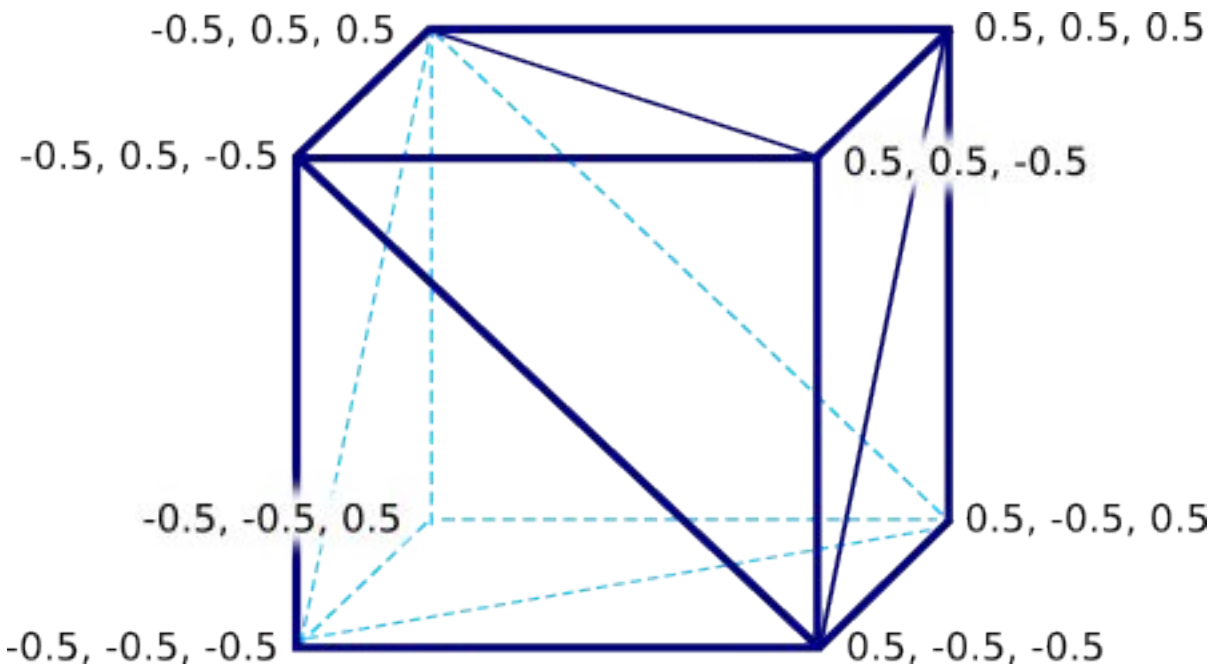
- Пусть p и q - кватернионы, тогда $pqv(pq)^{-1} = \mathrm{pqv}q^{-1}p^{-1}$
- Если задан кватернион q, то соответствующая ему матрица вращения равна

$$\begin{pmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{pmatrix}$$

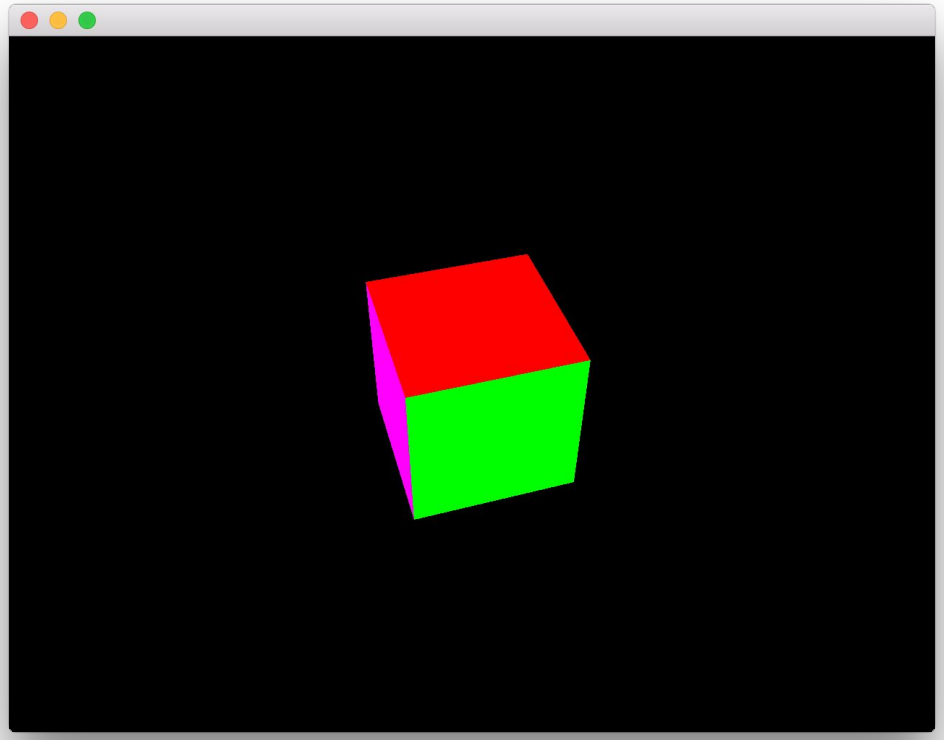
Рисуем куб

- Куб с ребром длиной 1 и центром в точке (0,0,0)
- Как нарисовать:
  1. Разбить все грани на 2 треугольника
  2. Перечислить координаты всех треугольников в правильном порядке

Рисуем куб

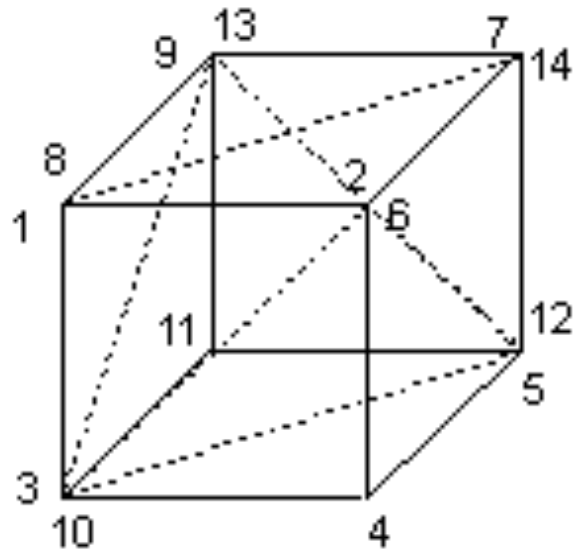


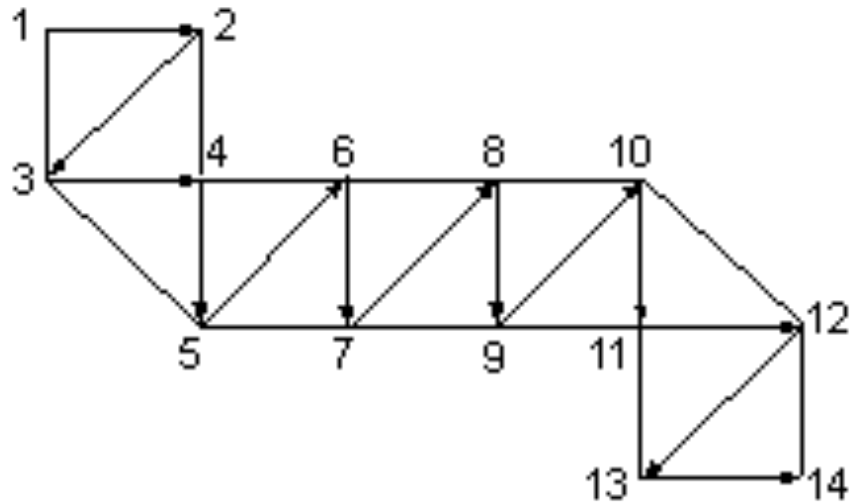
Рисуем куб: Вариант 1 – В лоб



```
void MainWindow::paintGL()
{
    // Очищает буфет цвета
    glClear(GL_COLOR_BUFFER_BIT);

    modelview.setToIdentity();
    modelview.translate(0.0, 0.0, -5.0);
    modelview.rotate(rotation);

    glMatrixMode(GL_PROJECTION);
    glLoadMatrixf((const GLfloat*)projection.data());
    glMatrixMode(GL_MODELVIEW);
    glLoadMatrixf((const GLfloat*)modelview.data());

    //Рисуем в режиме треугольников
    glBegin(GL_TRIANGLES);

    //Грань 1
    glColor3f(1, 0, 0);

    glVertex3f(-0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(-0.5f, 0.5f, 0.5f);
    glVertex3f(-0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    //Грань 2
    glColor3f(0, 1, 0);

    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, -0.5f);
```

Рисуем куб: Вариант 2 – Чуть умнее





Рисуем куб: Вариант 2 – Чуть умнее

- Используем режим GL_TRIANGLE_STRIP:
  - Пусть есть N вершин
  - Для четных n рисуется треугольник с вершинами n, n+1, n+2
  - Для нечетных n рисуется треугольник с вершинами n+1, n, n+2
  - В итоге рисуется N-2 треугольников

```
//Рисуем в режиме связанных треугольников
glBegin(GL_TRIANGLE_STRIP);

//Грань 1
glColor3f(1, 0, 0);

glVertex3f(-0.5f, -0.5f, 0.5f);
glVertex3f(0.5f, -0.5f, 0.5f);
glVertex3f(-0.5f, 0.5f, 0.5f);
glVertex3f(0.5f, 0.5f, 0.5f);
//Грань 2
glColor3f(0, 1, 0);

glVertex3f(0.5f, -0.5f, 0.5f);
glVertex3f(0.5f, -0.5f, -0.5f);
glVertex3f(0.5f, 0.5f, 0.5f);
glVertex3f(0.5f, 0.5f, -0.5f);
```