Гуиде на первую л/р по **компьютерной графике**

На чем будем писать: *на C++ под Windows (WinAPI)*

В чем будем писать: *в Visual Studio*

Нужны ли дополнительные инструменты и библиотеки: *для дальнейших лаб нам понадобится Blender, так же нужен будет графический редактор.*

О том, что такое Open GL прочитайте на Википедии (потом, дома).

В двух словах: мы будет OGL воспринимать как библиотеку функций для управления драйвером видеокарты, а видеокарта умет «рисовать» картинки. (на самом деле она считает хорошо, а в данном контексте рисовать == считать).

Вы писали программы до этого, должны были. В основном они использовали центральный процессор для получения результатов (каждая строчка кода так или иначе проходит через ЦПУ). Этот случай не исключение, наши вызовы ОпенГЛьных функций будут попадать сначала про процессор, а затем в драйвер видеокарты, и он уже скажет видеокарте, что ей делать. Схема работы, на самом деле, сложнее, но работает примерно вот так.

Начнем.

Все помнят обычный Hello, World на c++

```
#include <iostream>

int main()
{
	std::cout << "Hello, World";
	return 0;
}
```

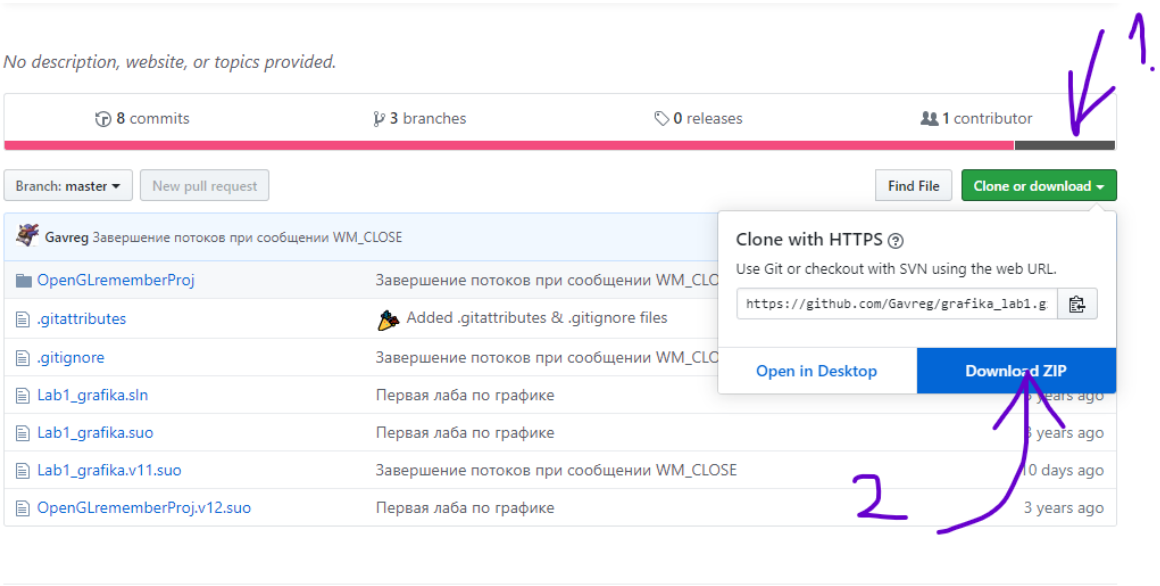
Создание ОпенГЛного аналога Hello, World задача более сложная, потому что:

1. У нас не консольное приложение, а оконное! Это подразумевает зависимость от платформы (Windows, Linux, …), так как оконная система без фреймворков (например Qt) уникальная для каждой системы.
2. C++ - нативный язык, и программы, написанные на нем исполняются операционной системой без посредников. Традиционно, графику программируют именно на C++, так как эта область требует очень высокого быстродействия от программ. Конечно, можно использовать OGL практически на любом языке, и он будет точно такой же, **с учетом особенностей синтаксиса языка и особенностей порта библиотеки OGL на этот язык**. Так что, все что мы с вами будем делать на этих лабах – работает везде где есть OGL – Linux, Android, PlayStation (если достанете DevKit, xexe), короче, на всем где есть 3D и оно не имеет связи Microsoft.
3. Инициализация пустого окна на чистом WinApi довольно рутинная задча, нужно вручную прописать довольно много параметров, да еще и таких, которые бы потом понравились OpenGL, запустить цикл перехвата сообщений, и.т.д. Не буду вдаваться в подробности, в Visual Studio есть для таких задач шаблонный проект.
4. В пустом окне надо проинициализировать OpenGL. Тоже требует терпения и прописывания определённых настроек.

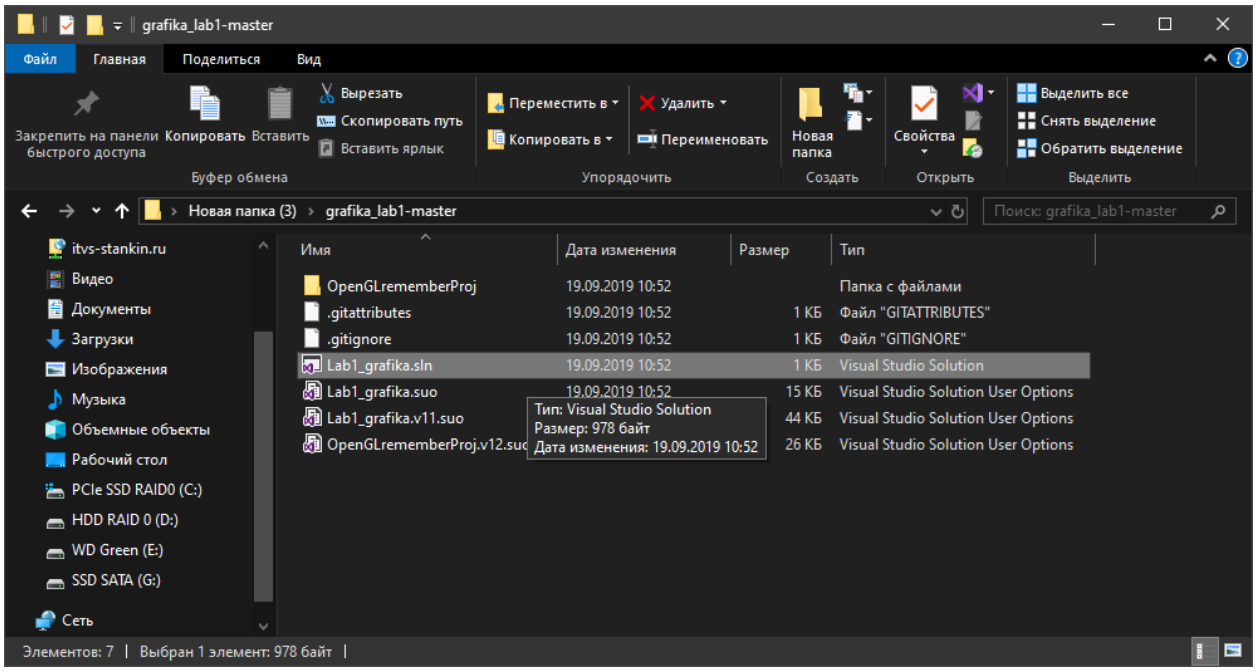
НО! Но все эти действия делать не надо.

Итак, погнали:

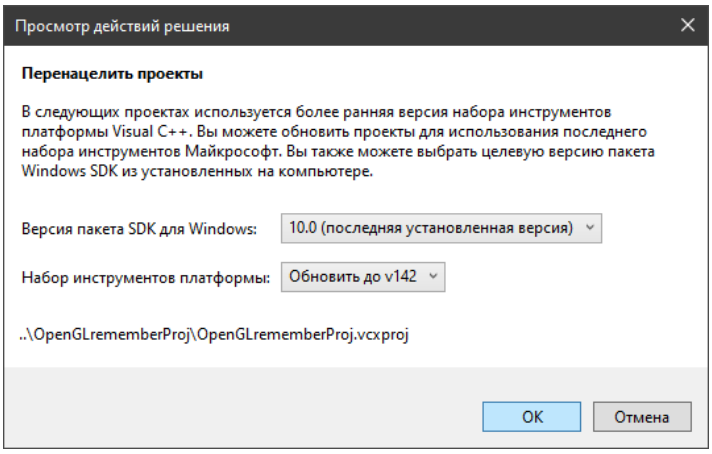
1. Качаем заготовку для лабы https://github.com/gavreg/grafika_lab1



2. Скачанный zip архив распаковываем, запускаем sln файл. (уже готовое решение с проектом) (вспоминая лекцию Лакуниной по открытию решения в Visual Studio)

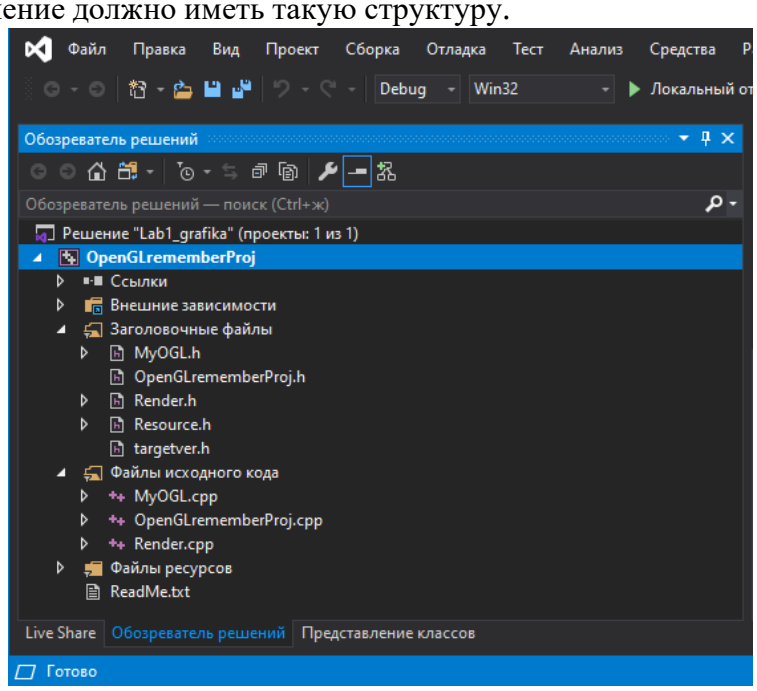


3. Этот проект я делал в 12й студии, если а классе установлена более новая версия, студия предложит обновить библиотеки для сборки - соглашаемся.

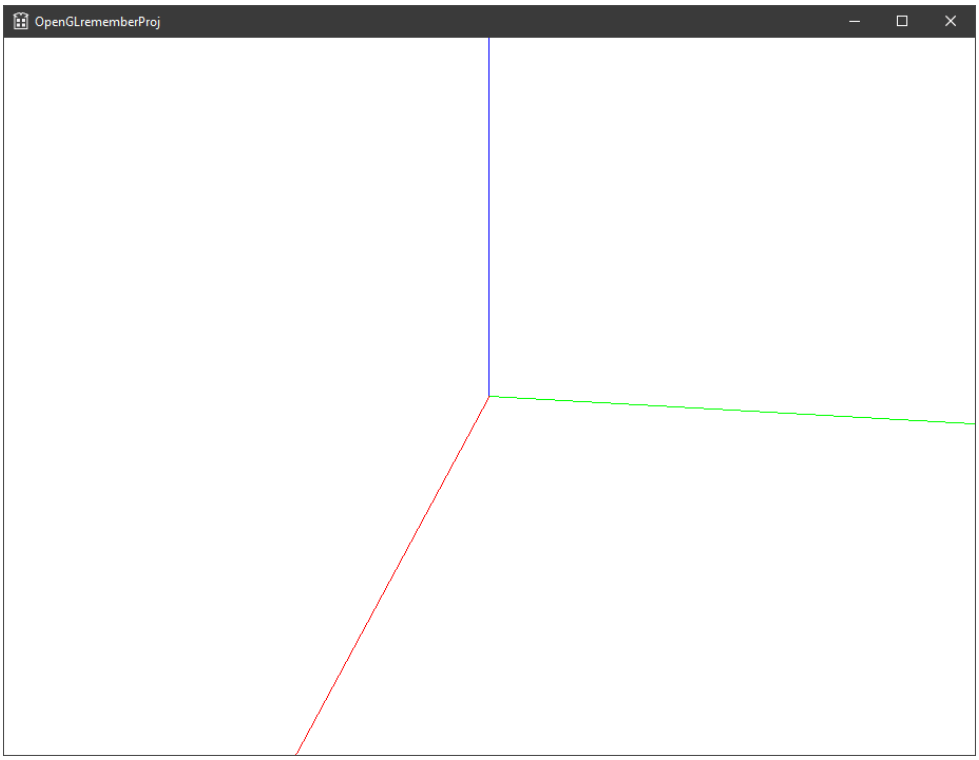


v142 – это инструменты VS 2019. У вас могут быть другие, зависит от версии студии.

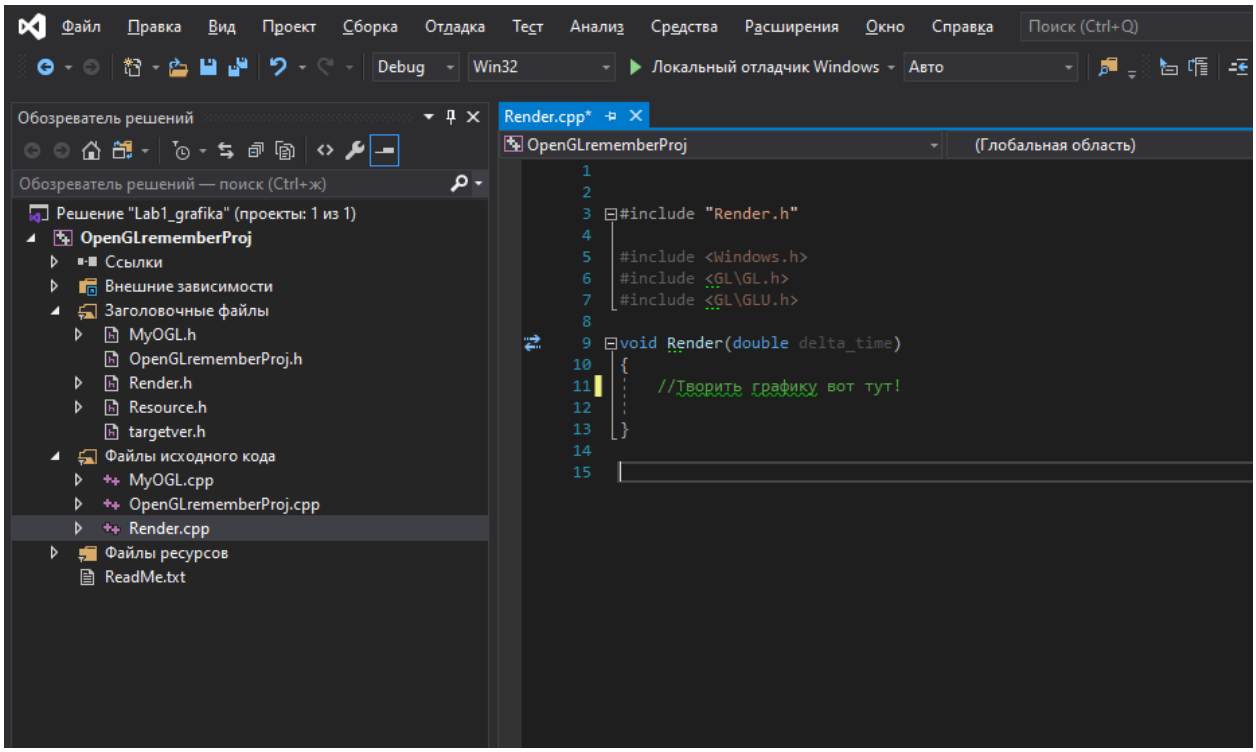
4. Открытое решение должно иметь такую структуру.



5. Попробуйте собрать проект(F5). Должно получиться что-то вроде этого.



6. Открываем файл Render.cpp и творим в нем КОМПЬЮТЕРНУЮ ГРАФИКУ



Все что будем делать, нужно писать внутри ф-ции render!

О там как эта «болванка» для лабы работает, откуда и почему вызывается ф-ция Render, можно будет спросить позже, сосредоточьтесь на задании.

Теперь начнем рисовать

1. Все рисование в OGL происходит между функциями glBegin и glEnd. Первая начинает рисование, вторая – заканчивает. После каждого бегина строго должен идти энд, их НЕЛЬЗЯ вкладывать друг в друга (да и незачем).

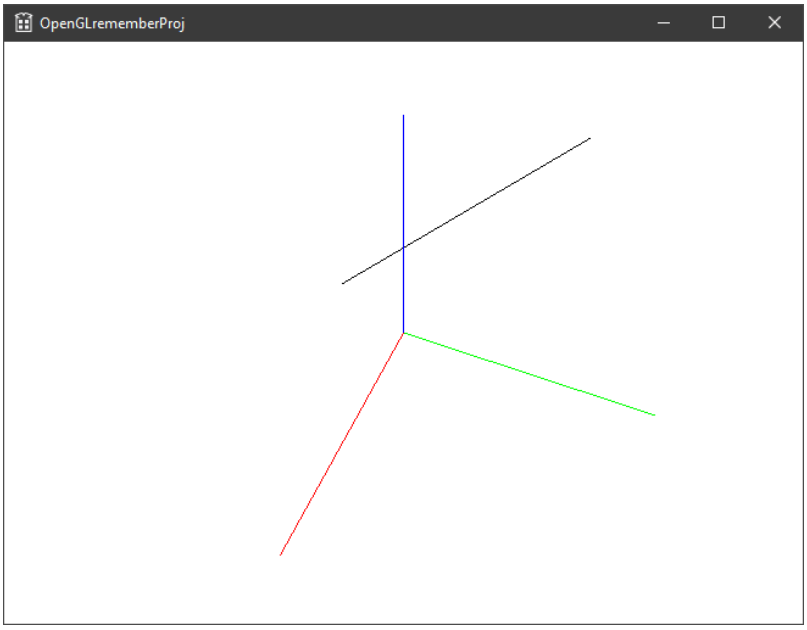
Так можно	Так нельзя
glBegin … glEnd glBegin … glEnd	glBegin … glBegin … glEnd … glEnd

2. Рисование заключается в том, чтобы «кормить» видеокарте точки в пространстве, а она их построит в соответствии с настройками блока glBegin и самого OLG.
   Точки видеокарте задаются функцией glVetrex3d, давайте посмотрим, почему она так называется:

3. **Задание 1. Рисуем линию.** Давайте попробуем нарисовать линию. Например, хочу нарисовать отрезок с точками (1,-2,3) и (-4, 5, 8)

```
glBegin(GL_LINES);
        glVertex3d(1, -2, 3);
        glVertex3d(-4, 5, 8);
glEnd();
```

Результат:



Как не трудно догадается функции glVertex3d нужны координаты точки – x, y, z.

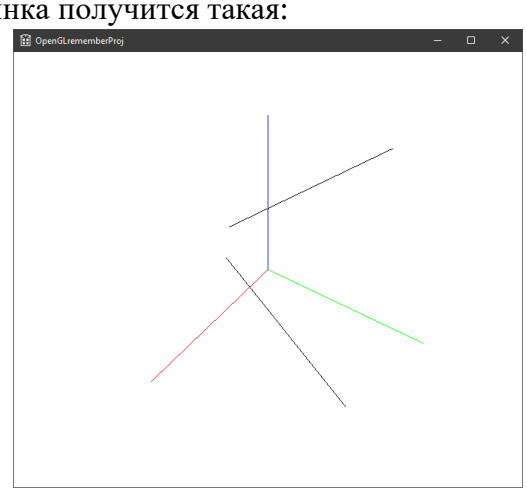
Чтобы ориентироваться в пространстве: оси x (красная, red), y (зеленая, green), z (синяя, blue) начинаются в точке (0,0,0) и имеют длину 10. Порядок запомнить легко: RGB – xyz.

4. **Задание 2. Рисуем две линии.** Если я хочу нарисовать вторую линию, я могу сделать это

так,	или так

`glBegin(GL_LINES);`
`  glVertex3d(1, -2, 3);`
`  glVertex3d(-4, 5, 8);`
 	
`glEnd();`
 	
`glBegin(GL_LINES);`
`  glVertex3d(-2, -5, -3);`
`  glVertex3d(7, 9, 2);`
`glEnd();`
	`glBegin(GL_LINES);`
	`  glVertex3d(1, -2, 3);`
	`  glVertex3d(-4, 5, 8);`
	
	`  glVertex3d(-2, -5, -3);`
	`  glVertex3d(7, 9, 2);`
	`glEnd();`

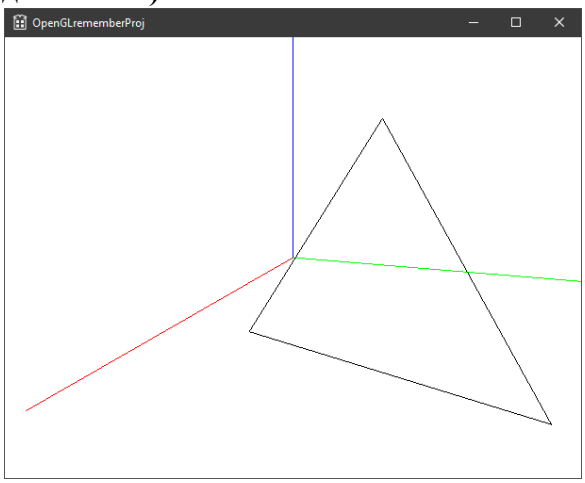
В обоих случаях картинка получится такая:



При этом второй случай более предпочтительный, потому что меньшее кол-во бегинов и эндов меньше нагружают видеокарту.
Вы можете заметить, что в режиме рисования GL_LINES, который мы указали в ф-ции glBegin, видеокарта берет по две точки последовательно и соединяет их линиями – 1 и 2, 3 и 4, 5 и 6, итд.

Ради пресечения потенциальных ошибок, посмотрите, как программа ведет себя если забыть закрыть какой ни будь glBegin.

Задание 3: вооружившись вышенаписанной информацией, постройте треугольник (у каждого свой!):



Напоминаю: мы пишем программу на C++, по этому все что вы знаете и умеете тут применимо. Циклы, указатели, итд. Дальше без этого – никак.

Например, более простое задание точки с помощью массива:

```
double A[] = { -1,-2,-3 };
…
glVertex3dv(A); //обратите внимание на букву v в фц-ии
                //Она говорит о том, что те самые 3 аргумента типа
                //double передаются в массиве (по указателю)
```

В дальнейших примерах буду так рисовать.

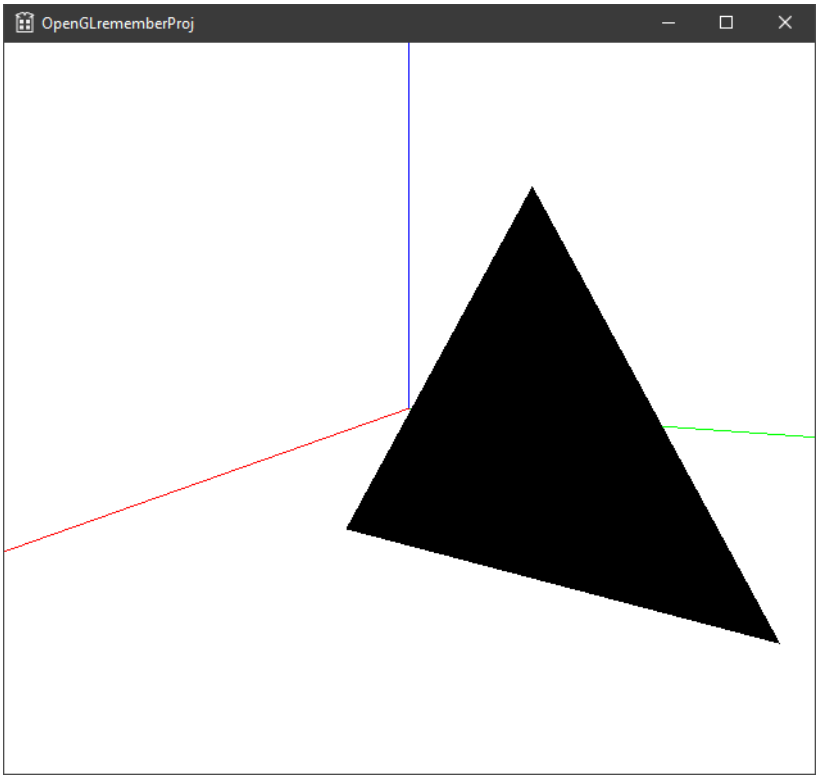
Массив можно инициализировать в любом месте программы, а вот функция glVertex работает только между бегином и эндом.

5. **Задание 4. Нарисовать треугольный полигон**. Теперь, давайте нарисуем полигон. Для этого нужно Begin переключить в другой режим рисования:

```
glBegin(GL_TRIANGLES);

glEnd();
```

Он работает точно так же, но «кушает» уже по **три** точки, соединяя их треугольником. (вспоминаем аксиомы стереометрии)

```
double A[] = { -1,-2,-3 };
double B[] = { 4, 7, -2 };
double C[] = { 2, 3, 4 };

glBegin(GL_TRIANGLES);
       glVertex3dv(A);
       glVertex3dv(B);
       glVertex3dv(C);
glEnd();
```



6. Что то невзрачный у нас треугольник, давайте покрасим его. Красить будем функцией glColor3d. Как видим из названия, она принимает три аргумента типа double – это три цветковые компоненты в формате RGB. **НО!** В отличии от привычного задания цвета целочисленным диапазоном [0..255], тут используется вещественный диапазон [0..1]

Некоторые простые цвета:

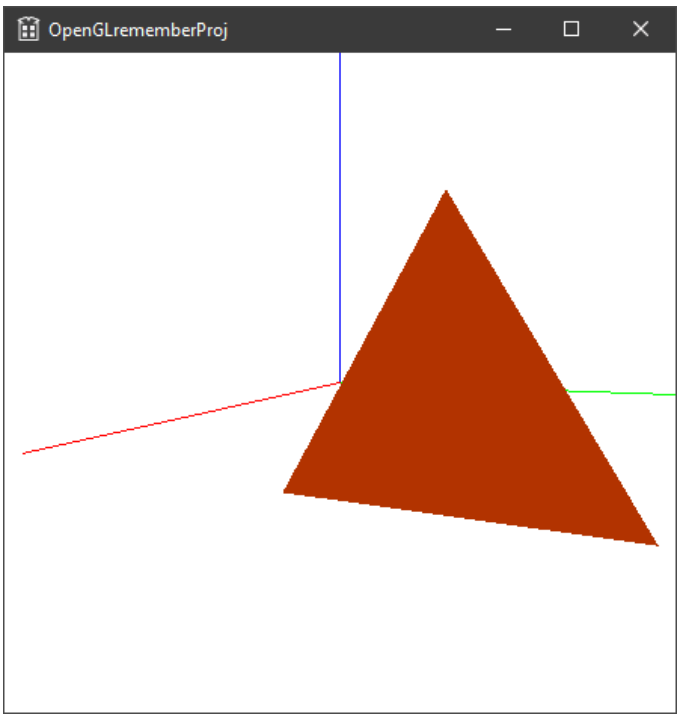
glColor3d(1,0,0)	красный
glColor3d(0,1,0)	зеленый
glColor3d(0,0,1)	синий
glColor3d(x,x,x)	x = [0..1] оттенки серого
glColor3d(1,1,1)	белый (самый светлый серый)
glColor3d(0,0,0)	черный (самый темный серый)

Задание 5. Закрасить треугольник.

Вот как раскрасил треугольник.

Функция glColor красит все, что идет ниже нее, до того момента, пока не дойдет до другой glColor. glColor может быть как внутри блока glBegin-glEnd, так и вне его.

```
double A[] = { -1,-2,-3 };
double B[] = { 4, 7, -2 };
double C[] = { 2, 3, 4 };

glBegin(GL_TRIANGLES);

	glColor3d(0.7, 0.2, 0);

	glVertex3dv(A);
	glVertex3dv(B);
	glVertex3dv(C);

glEnd();
```

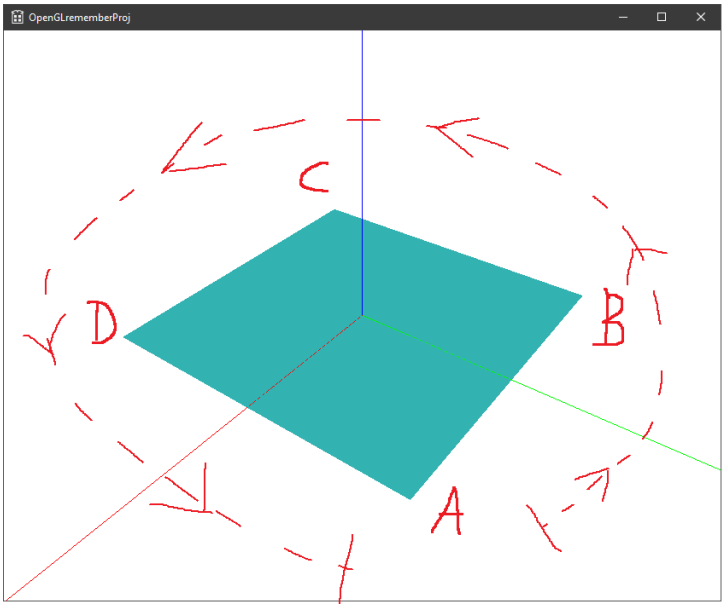


Стоит заметить: glColor красит только вершины. Если все вершины одного цвета, то и грань, которая их объединяет будет того же цвета. А вот, если все вершины имеют разные цвета, получается красиво. Попробуйте.

7. **Задание 5. Нарисовать квадратный полигон**. Последний примитив на сегодня – четырехугольник. Все как с треугольником, но «кушает» по 4 точки.

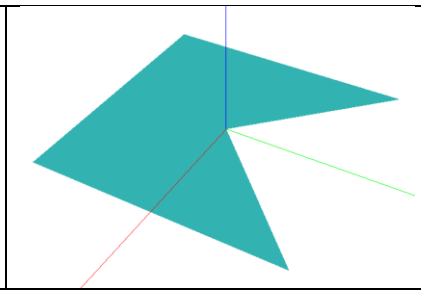
```
double A[] = { 1, 1, 0 };
double B[] = {-1, 1, 0 };
double C[] = {-1,-1, 0 };
double D[] = { 1,-1, 0 };

glBegin(GL_QUADS);

    glColor3d(0.2, 0.7, 0.7);

    glVertex3dv(A);
    glVertex3dv(B);
    glVertex3dv(C);
    glVertex3dv(D);

glEnd();
```

В отличии от треугольника, тут важен порядок точек, они должны быть заданы в порядке обхода четырехугольника по контуру.



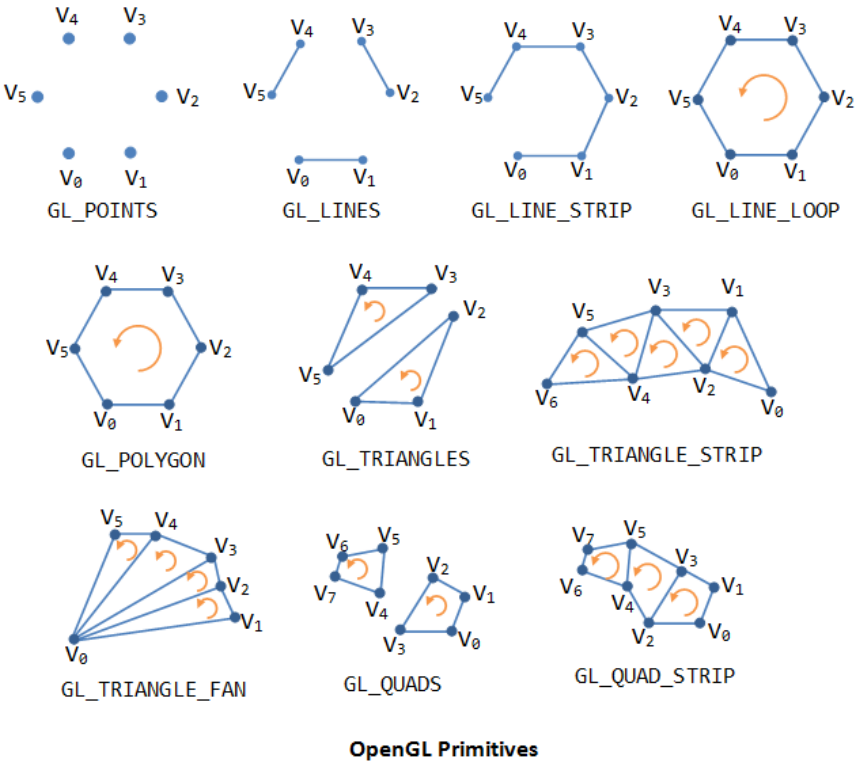
Вот что будет при неправильном порядке:

```
glVertex3dv(A);
glVertex3dv(C);
glVertex3dv(B);
glVertex3dv(D);
```



Помимо порядка задания точек, нужно БЫТЬ УВЕРЕННЫМ в том, что точки лежат в одной плоскости. В примере это очевидно, так как у всех точек аппликата (координата z) равна нулю. В более общем случае, когда точки случайно разбросаны в пространстве (попробуйте как нить повернуть квадрат вокруг оси X), сходу определить принадлежность точек плоскости нельзя. По этому в основном рисуют треугольниками.

Рассмотренные режимы рисования не единственные в openGL. Их довольно много:

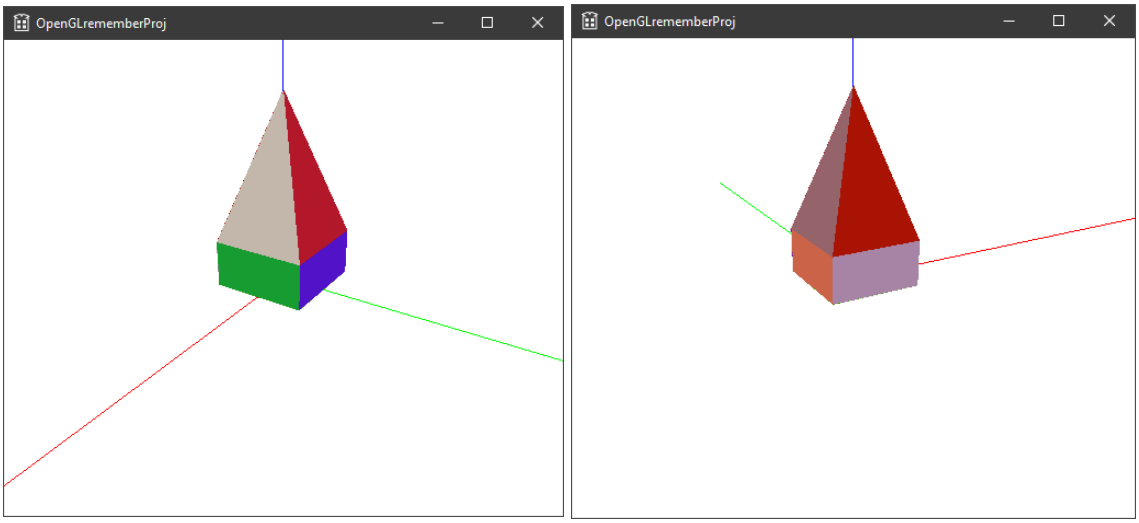


OpenGL Primitives

8. Давайте на практике закрепим знания:

ЗАДАНИЕ 6

Нарисовать вот это чудо. (Обязательно с центром в начале координат)

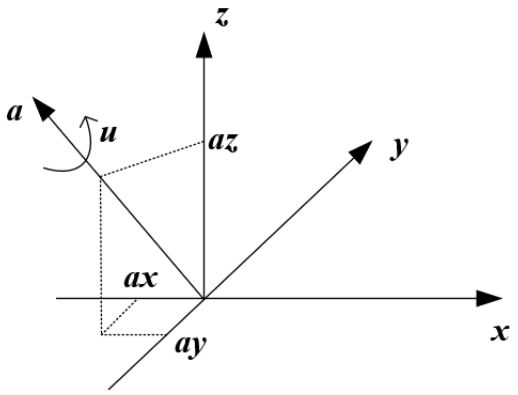


9. Повороты, перемещения и масштабирования.

Три функции управляют моделью:

glTranslated (x,y,z) – перемещает сцену на вектор [x,y,z].

glRotated(angle,ax,ay,az), поворачивает сцену на угол u в градусах, вокруг оси a, заданной вектором [ax,ay,az], идущего из начала координат.

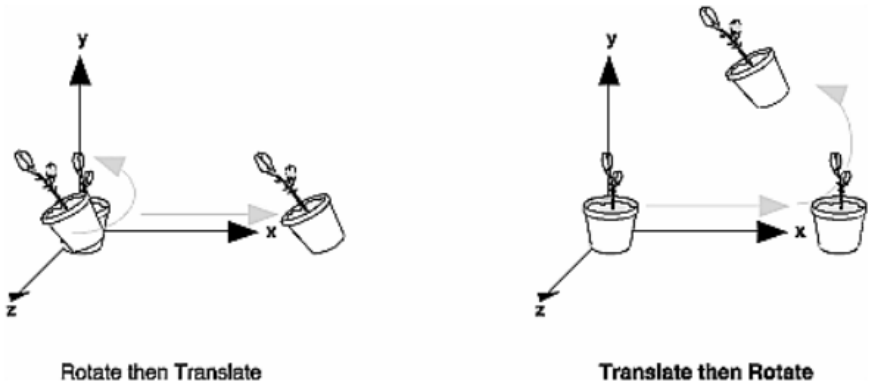


glScaled(x, y, z), масштабирует сцену по осям, фактически умножает все точки на x, y, z.

Функции применяются ко всему, что рисуется ниже их вызова.

Не очевидно, но факт: из за особенностей математического аппарата, функции выполняются в обратном порядке относительно их вызова, т.е. если вы написали Rotate, Translate – OGL сначала переместит, а потом повернет. (В дальнейшем, когда будет говорится о прядке преобразований, будет имеется ввиду порядок их применения, а не порядок команд кода)

Задание 7. Попробуйте применить их к своей фигуре и понять, как они работают. Попробуйте комбинации перемещение-поворот и поворот-перемещение.

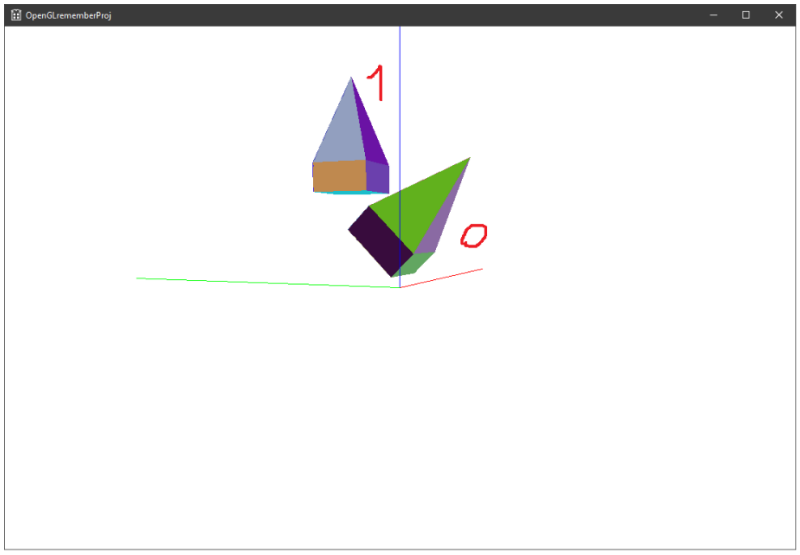


10. Если вы рисуете несколько объектов, и у каждого свои переносы-повороты, то чтобы они друг на друга не влияли их нужно изолировать парой функций:

    glPushMatrix();
    //тут перемещения, повороты, рисование фигур

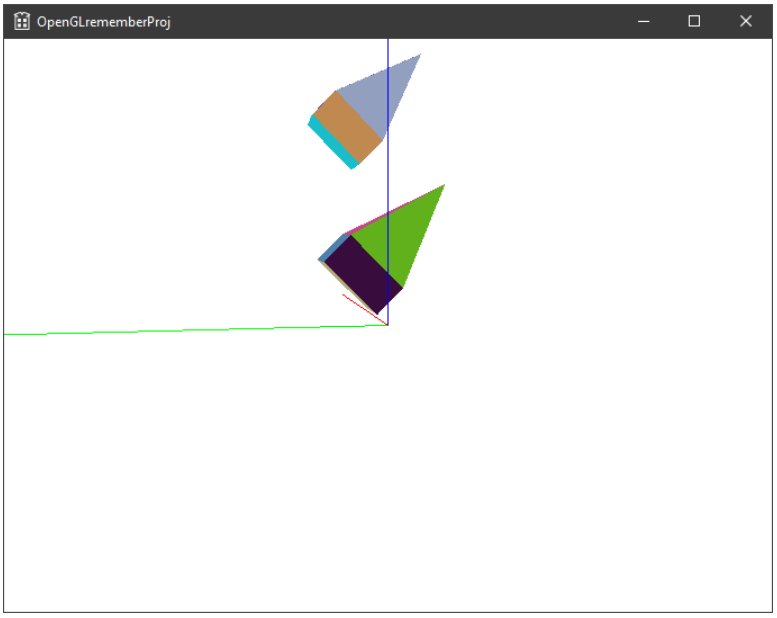
glPopMatrrix();

```
        glPushMatrix();
        glTranslated(1, 1, 1);
        glRotated(45, 1, 0, 0);
        figure(0); //перенес рисование фигуры в отдельную функцию
        glPopMatrix();

        glPushMatrix();
        glTranslated(3, 3, 3);
        figure(1);
        glPopMatrix();
```

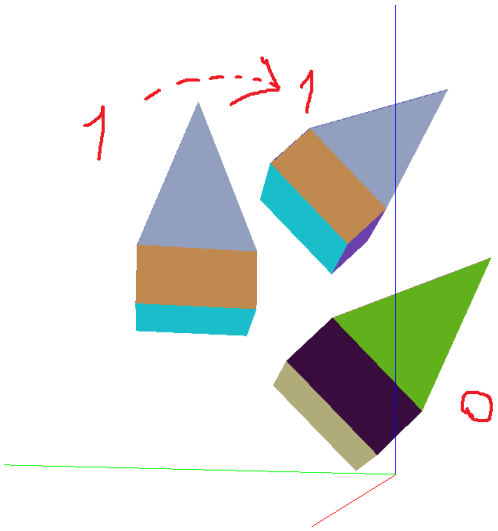


В примере фигура под номером 0 сначала была повернута на 45* вокруг оси x, затем перемещена на 1,1,1. Вторая фигура была просто перемещена на 3,3,3.

Если бы мы убрали все Пуши-Попы получилось бы так



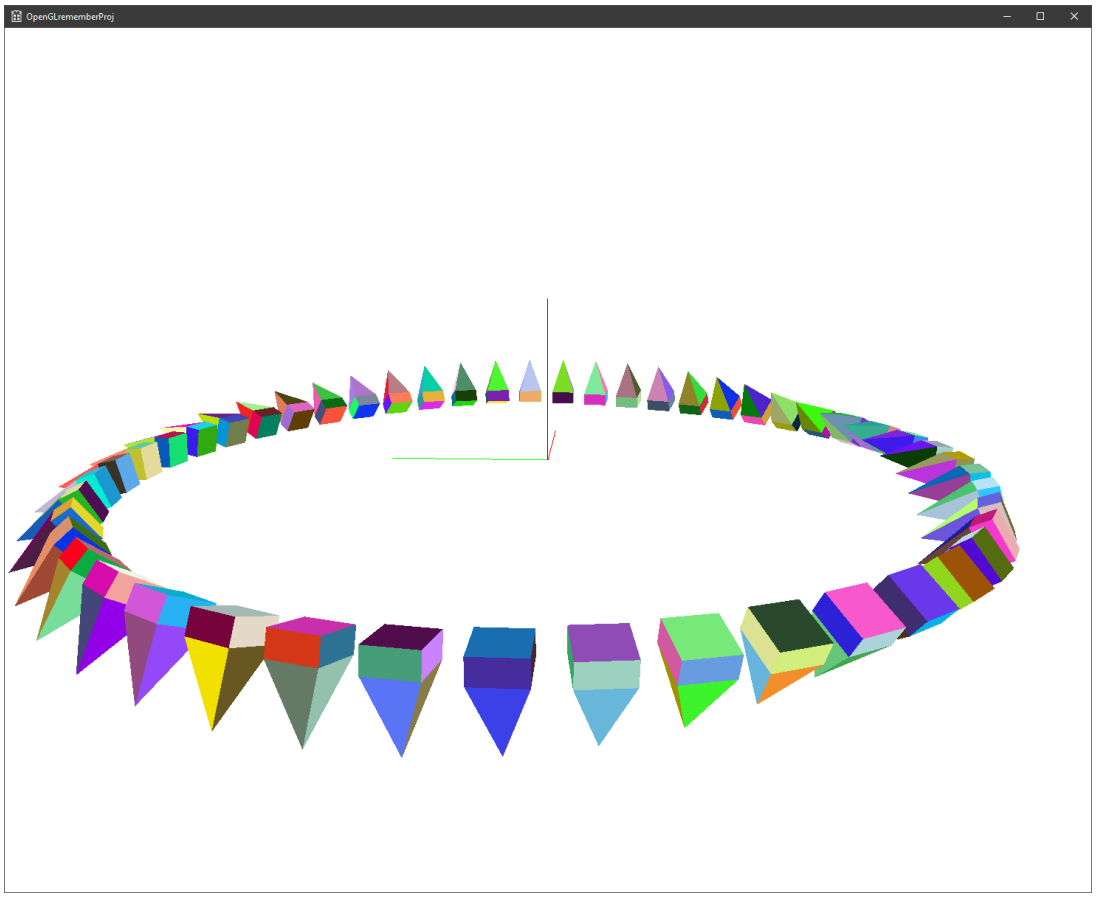
На первую фигуру подействовали бы только поворот и перемещение на [1, 1, 1], а вот на вторую сначала перемещение на [3,3,3], потом поворот на 45 и перемещение на [1, 1, 1].

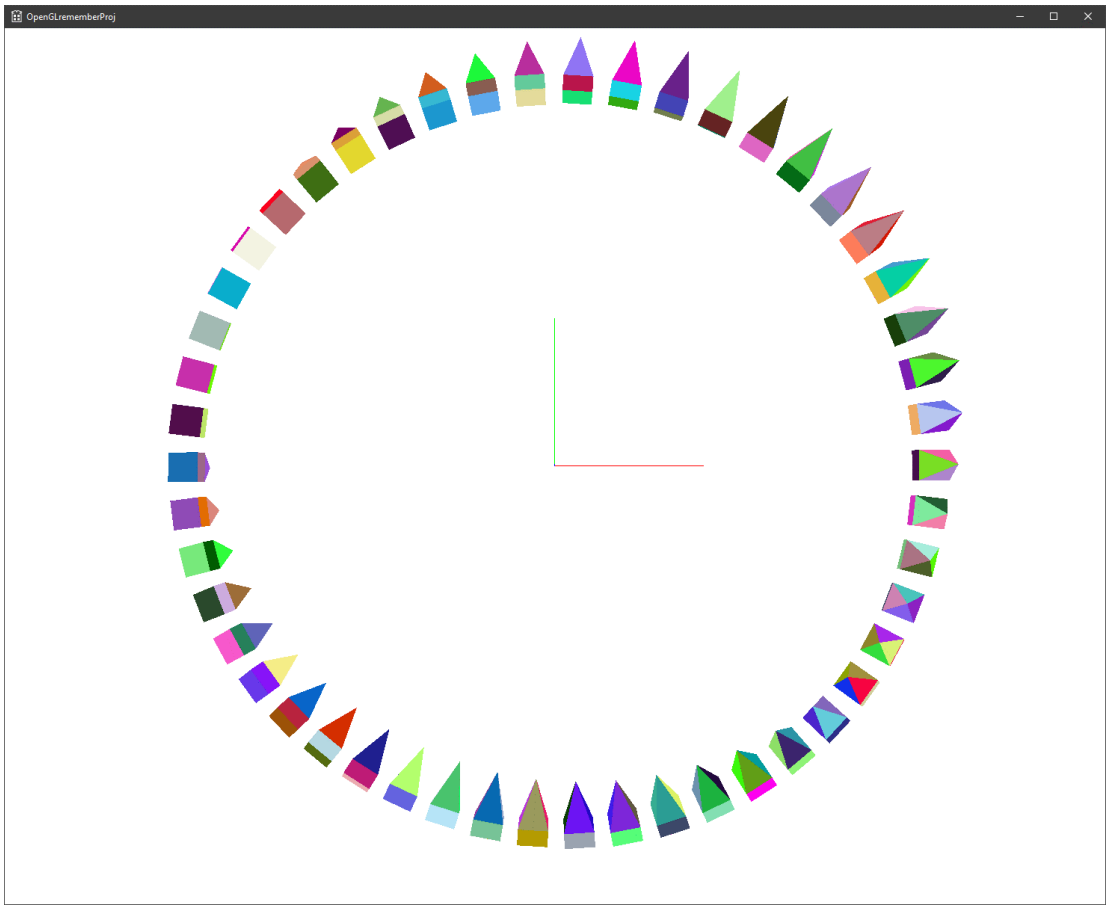


Вот так сместили вторую фигуру удаление пушей и попов.

ЗАДАНИЕ 8! Финальное (но это только разминка!)

Комбинацией ротейтов и транслейтов сделать такое





(тут 50 фигурок)

Задание 9. Ммм справились? Неплохо.

Попробуйте сделать вот это https://youtu.be/4LNw892WFU0

Подсказка для создания анимации - фунция Render() постоянно вызывается в цикле, перерисовывая картинку, и параметр delta_time на входе – кол-во времени с предыдущего рендера в СЕКУНДАХ. (именно поэтому вы ведете движение камеры, что картинка постоянно перерисовывается)

ЗАДАНИЕ 10. ДОШЛИ ДО СЮДА??

Тогда.. закрепим полученные знания и сделаем что ни будь с анимацией..

https://youtu.be/2Zd8SXo6Szs