Компьютерная графика Лекция 0

Краткое введение в курс

Определение

- **«Компью́терная гра́фика** (также **маши́нная графика**) — область деятельности, в которой компьютеры наряду со специальным программным обеспечением используются в качестве инструмента, как для создания (синтеза) и редактирования изображений, так и для оцифровки визуальной информации, полученной из реального мира, с целью дальнейшей её обработки и хранения.»

https://ru.wikipedia.org/wiki/Компьютерная_графика

Задачи и инструменты

Задачи	Приложения	Инструменты разработчика
Формирование трёхмерных моделей и сцен	Blender, Maya, 3ds Max, …	
Рендеринг	SOLIDWORKS Visualization, V-Ray, …	OpenGL, DirectX, VTK, …
Редактирование изображений	Photoshop, GIMP, …	OpenCV, ITK, …

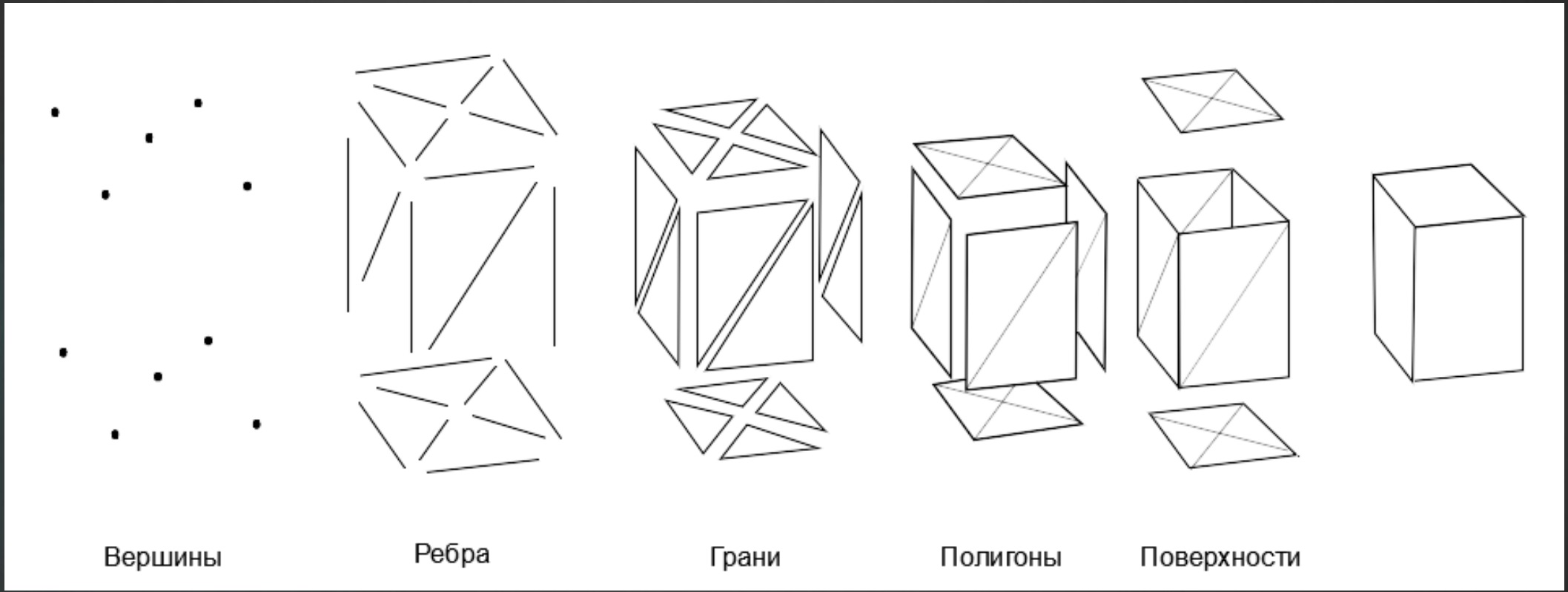
Цель курса

- В этом курсе мы изучим принципы трёхмерной визуализации. В течение курса мы разработаем программный модуль, реализующий базовые функции визуализации.
- *В этом курсе мы не будем писать приложения с использованием OpenGL.*

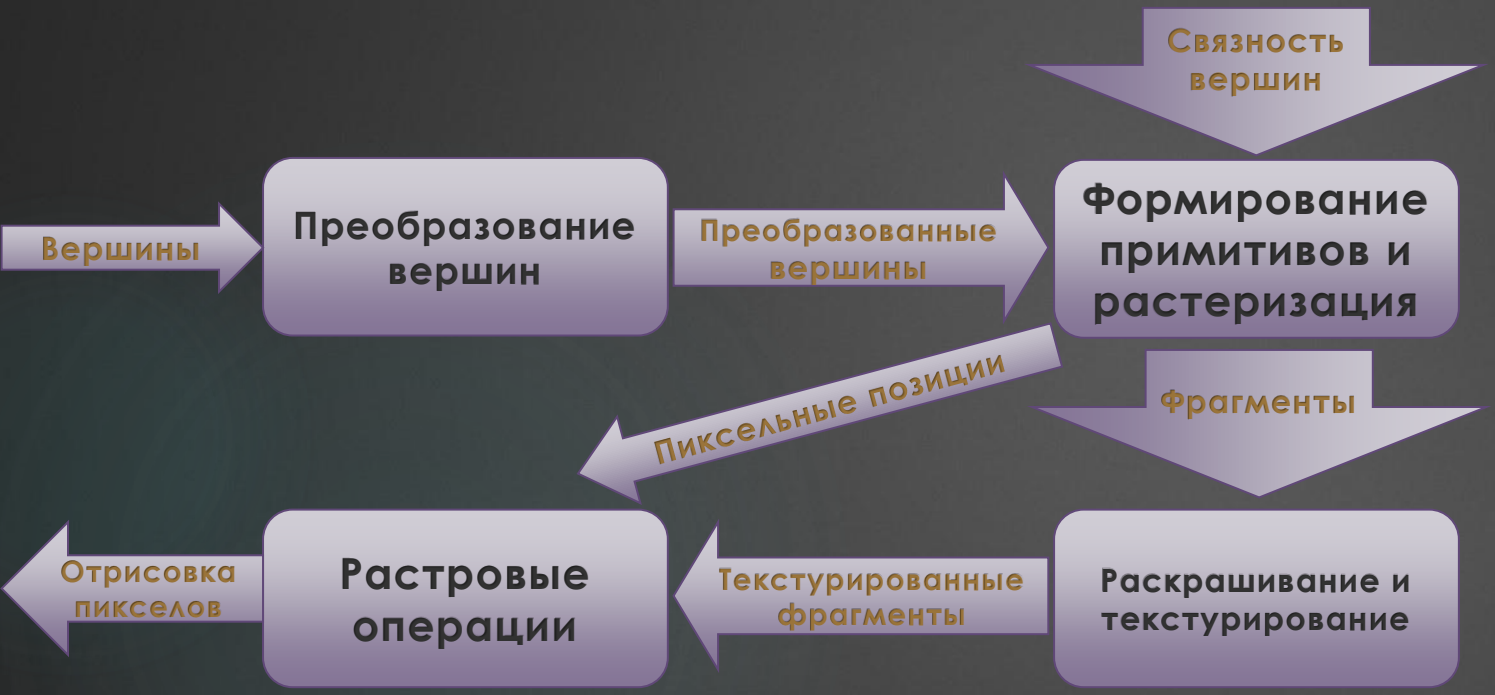
OpenGL

- **OpenGL** (**Open G**raphics **L**ibrary) — спецификация, определяющая платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.
- Основным принципом работы OpenGL является получение наборов векторных графических примитивов в виде точек, линий и треугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти. Векторные трансформации и растеризация выполняются графическим конвейером (graphics pipeline). Абсолютное большинство команд OpenGL попадает в одну из двух групп: либо они добавляют графические примитивы на вход в конвейер, либо конфигурируют конвейер на различное исполнение трансформаций.

Полигональная модель

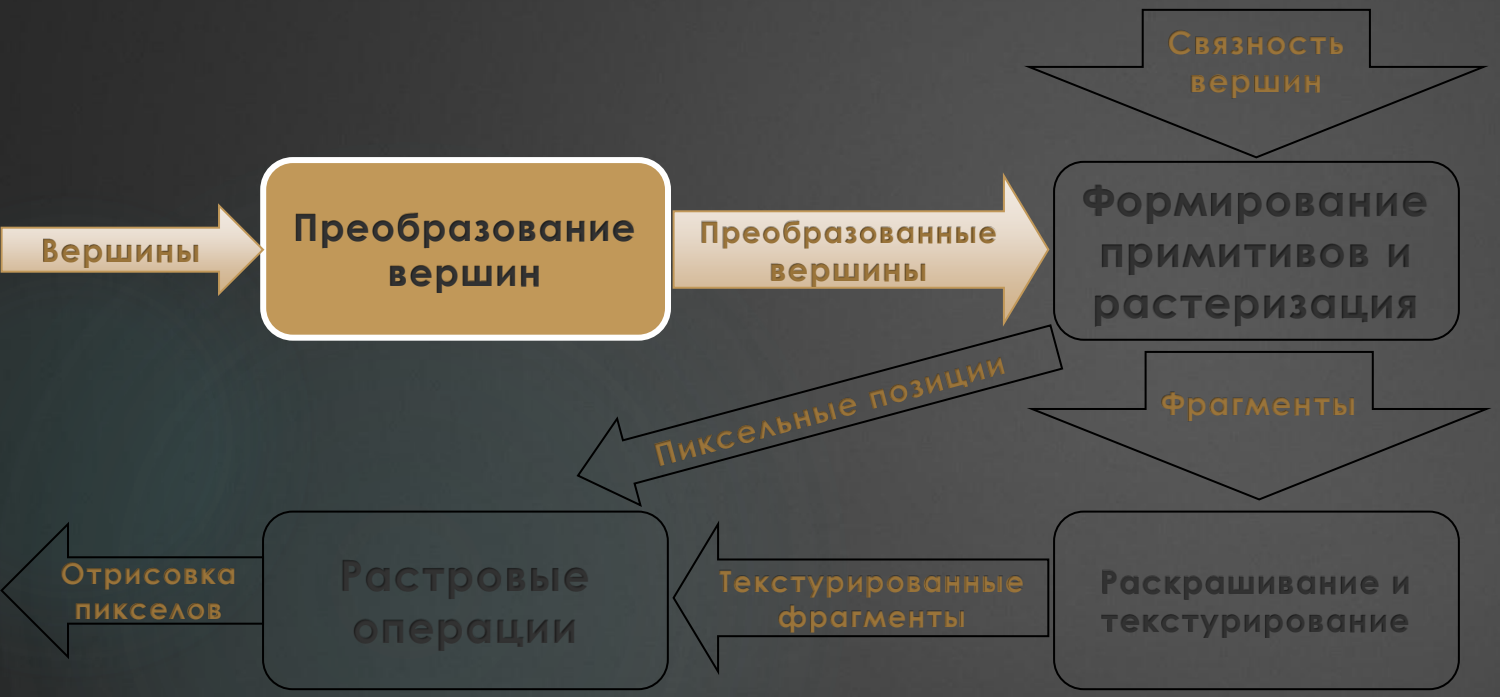


Графический конвейер



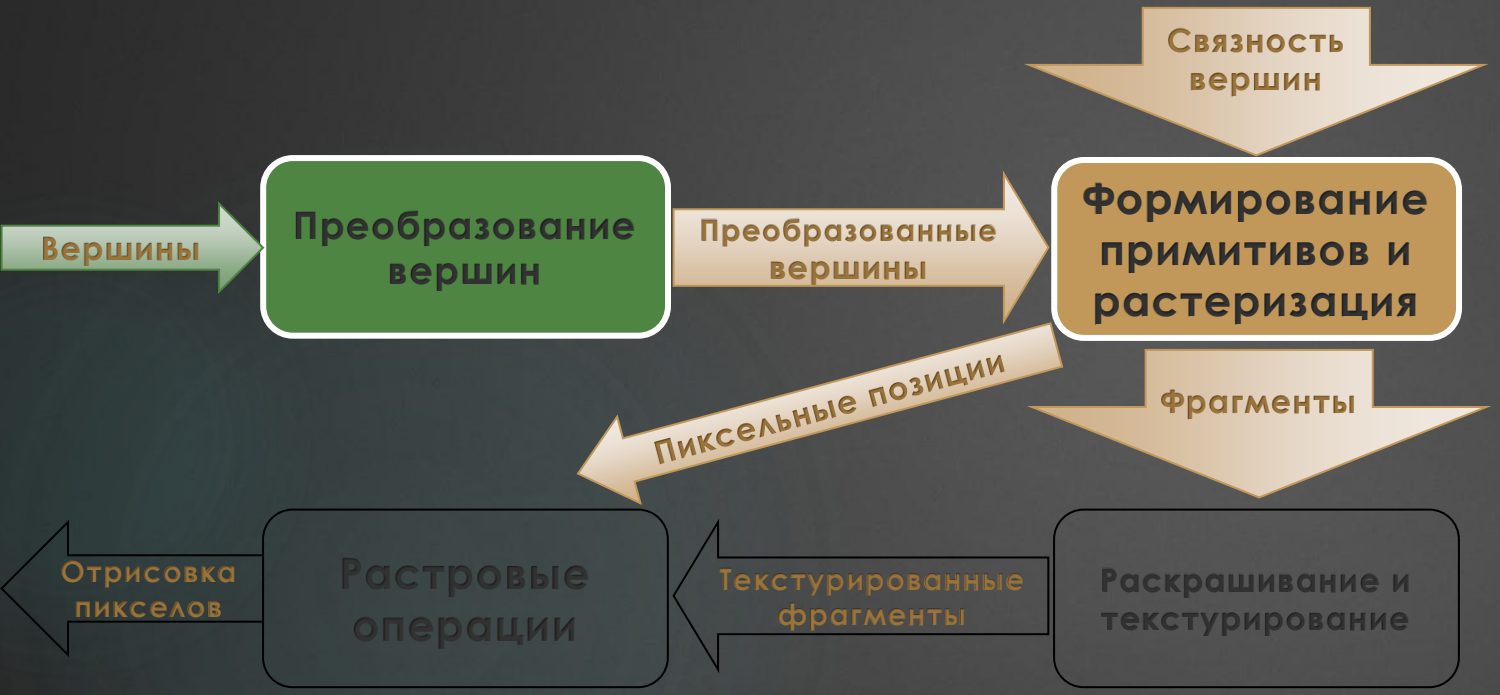
- Преобразование вершин:
  - На входе: координаты, цвет, нормаль, текстурные координаты
  - Трансформация позиции, вычисление освещённости, генерация текстурных координат
- Формирование примитивов и растеризация:
  - На входе: трансформированные вершины и информация о примитивах (полигонах)
  - Вычисление фрагментов и пиксельных позиций примитивов
- Раскрашивание и текстурирование:
  - Входные данные: информация о фрагментах
  - Наложение текстуры, цвета, эффектов
- Растровые операции:
  - Входные данные: положение и значение цвета пиксела
  - Тестирование на необходимость отрисовки

Преобразование вершин



- Преобразования, связанные с положением, ориентацией в пространстве и размером объектов, а также положением и ориентацией в пространстве камеры.

Формирование примитивов и растеризация



- Определение соответствующих пикселей для каждой грани объектов сцены.
- Вычисление пространственных координат для каждого пикселя грани

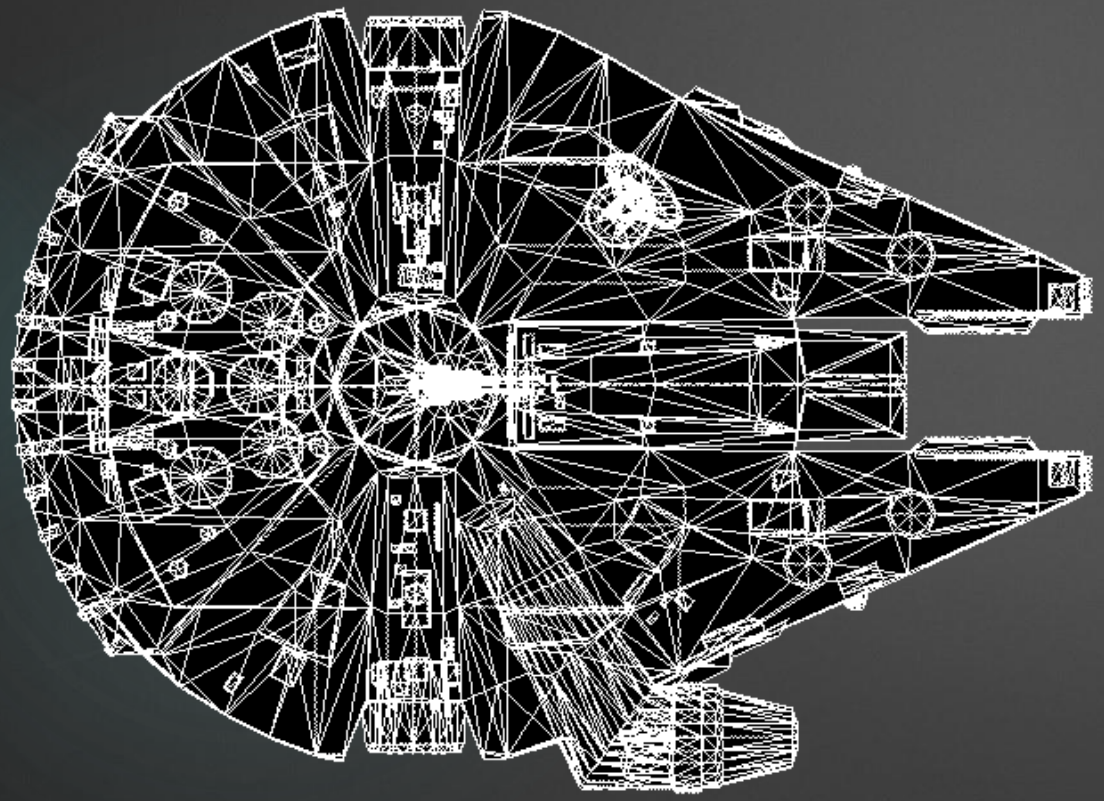
Графический конвейер

- Определение цвета каждого пикселя, в том числе, зависящего от текстуры, освещённости, теней, отражений и других параметров сцены и объектов.

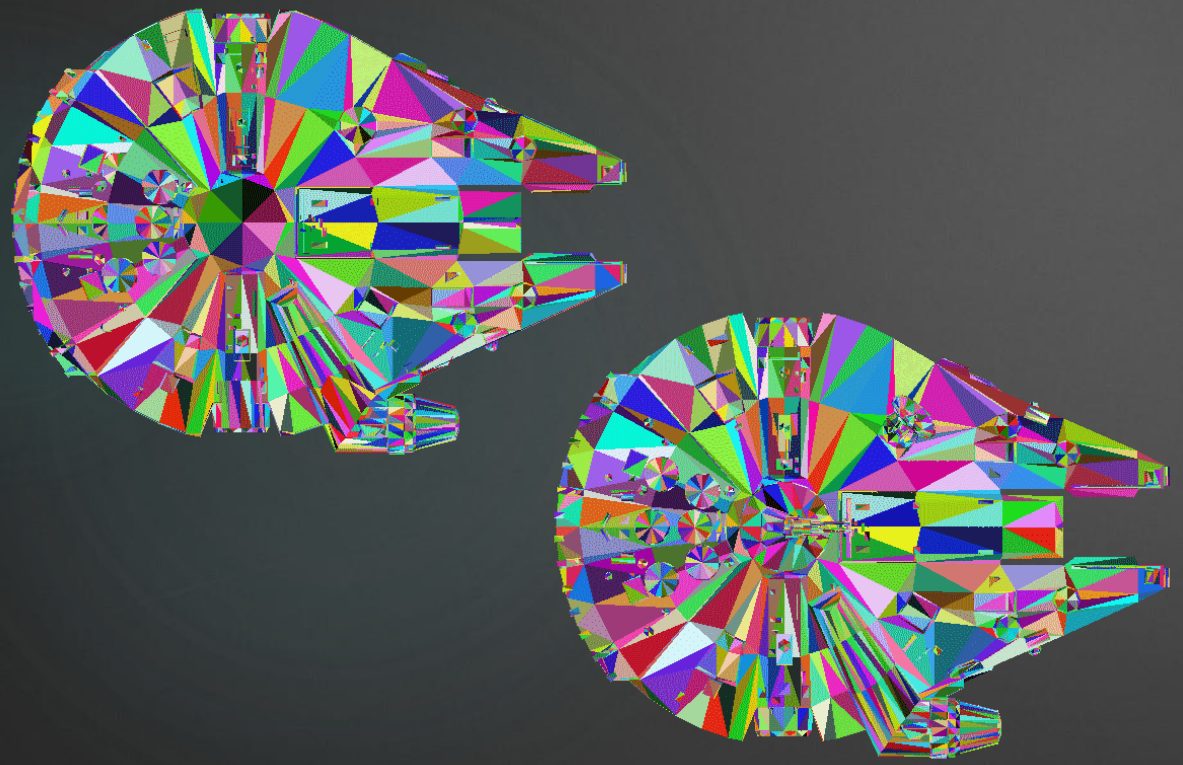
Графический конвейер

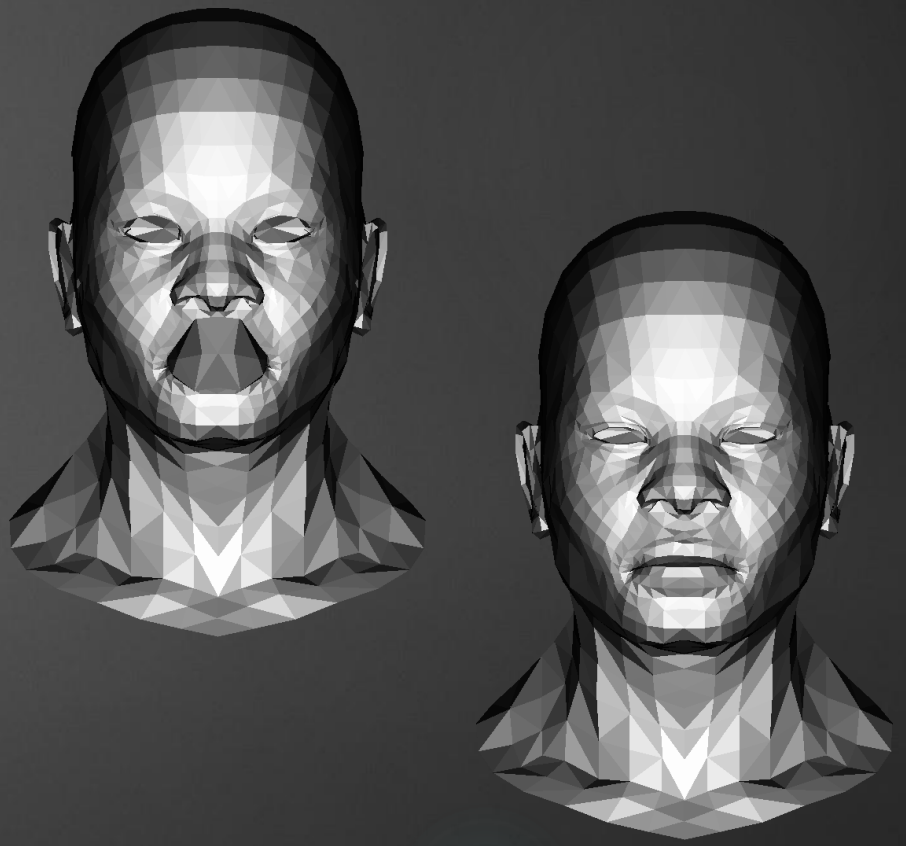
- Определение необходимости отрисовки каждого пикселя в зависимости от их трёхмерных координат и параметров сцены.

Загрузка модели из файла. Растеризация прямых линий.



Растеризация треугольников. Удаление невидимых поверхностей (z-буфер).





Перспективные искажения. Перемещение и поворот камеры.

Освещение. Текстуры. Тени.

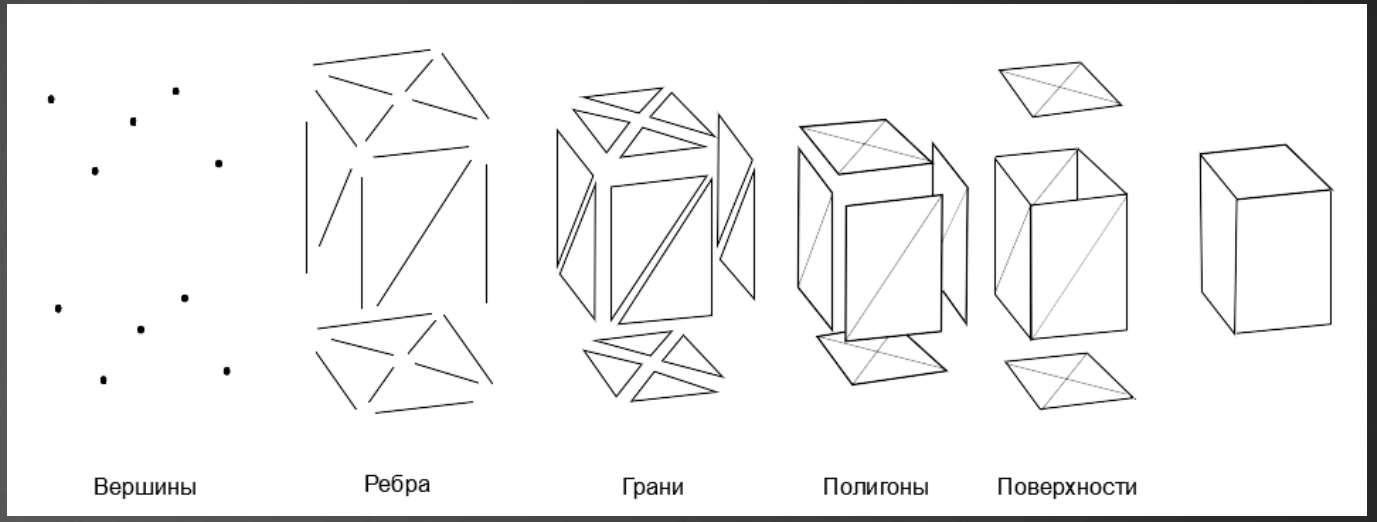
Языки и библиотеки.

- Python:
  - Pillow (раньше называлась PIL, Python Imaging Library)
- C++:
  - LodePNG,
  - LibPNG,
  - FreeImage,
  - CImg,
  - ~~OpenCV~~
- Java и любые другие:
  - ???

Спецификация OBJ-файла.

Общая структура

- mtllib [имя внешнего MTL файла]
- …
- v 0.123 0.234 0.345 1.0
- v …
- …
- vt 0.500 -1.352 [0.234]
- vt …
- …
- vn 0.707 0.000 0.707
- vn …
- …
- f 1 2 3
- f 3/1 4/2 5/3
- f 6/4/1 3/5/3 7/6/5
- f 6//1 3//3 7//5
- f …
- …
- # Группа
- g Group1
- …
- # Объект
- o Object1



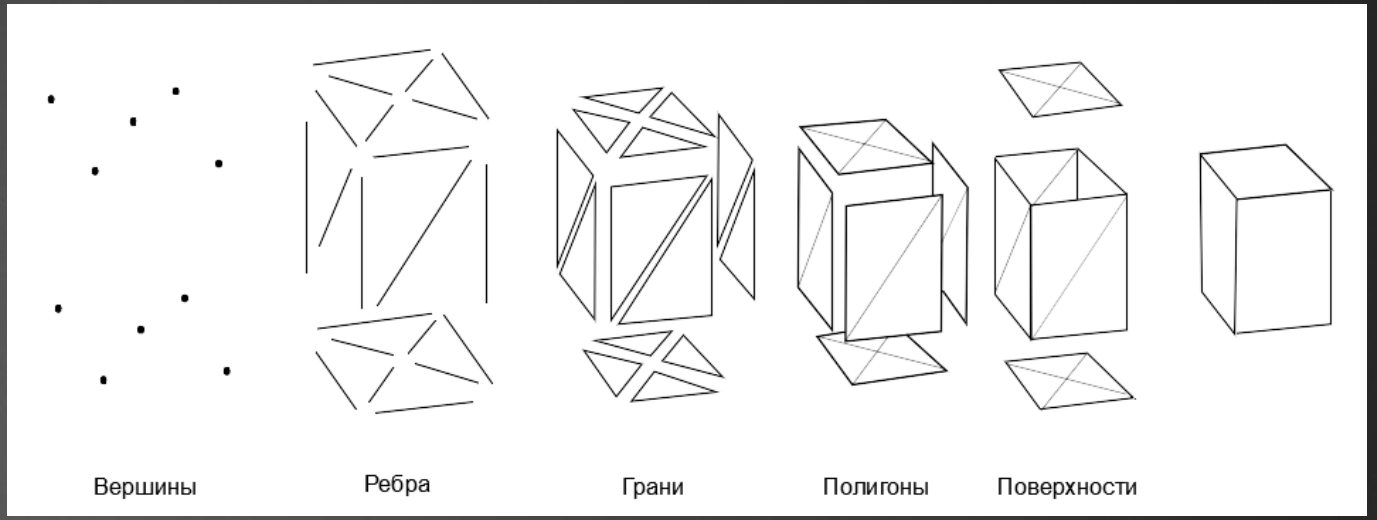
Спецификация OBJ-файла.

Индексация вершин

- # Список вершин, с координатами (x,y,z).
- v 0.123 0.234 0.345
- v ...
- ...

- # Текстурные координаты (u,v).
- vt 0.500 -1.352 [0.234]
- vt ...
- ...

- # Нормали (x,y,z); нормали могут быть не нормированными.
- vn 0.707 0.000 0.707
- vn ...
- ...



Спецификация OBJ-файла.

Вершины

- Каждая поверхность (полигон) может состоять из трех или более вершин.
- f v1 v2 v3 v4 ...
- Индексация начинается с первого элемента, а не с нулевого как принято в некоторых языках программирования, также индексация может быть отрицательной. Отрицательный индекс указывает позицию относительно последнего элемента (индекс -1 указывает на последний элемент).
- Наряду с вершинами могут сохраняться соответствующие индексы текстурных координат.
- f v1/vt1 v2/vt2 v3/vt3 v4/vt4 ...
- Также допустимо сохранение соответствующих индексов нормалей.
- f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 v4/vt4/vn4 ...
- При отсутствии данных о текстурных координатах допустима запись с пропуском индексов текстур.
- f v1//vn1 v2//vn2 v3//vn3 v4//vn4 ...

Спецификация OBJ-файла.

Файл материалов

- Информация о простых материалах в файле выглядит следующим образом:

- newmtl название_материала1 # Объявление очередного материала
- # Цвета
- Ka 1.000 1.000 0.000 # Цвет окружающего освещения
- Kd 1.000 1.000 1.000 # Диффузный цвет
- # Параметры отражения
- Ks 0.000 0.000 0.000 # Цвет зеркального отражения (0;0;0 - выключен)
- Ns 10.000 # Коэффициент зеркального отражения (от 0 до 1000)
- # Параметры прозрачности
- d 0.9 # Прозрачность указывается с помощью директивы d
- #Следующий материал
- newmtl название_материала2
- ...

Спецификация OBJ-файла.

Файл материалов

- Информация о материалах, содержащих текстуры, выглядит следующим образом:

- newmtl Название материала
- Ka 1.000 1.000 1.000
- Kd 1.000 1.000 1.000
- Ks 0.000 0.000 0.000
- d 1.0
- map_Ka Название_ambient_текстуры.tga
- map_Kd Название_diffuse_текстуры.tga
- map_Ks Название_specular_текстуры.tga

- map_bump Название_текстуры_нормали.tga

Компьютерная графика Модуль 1

Растеризация прямых линий

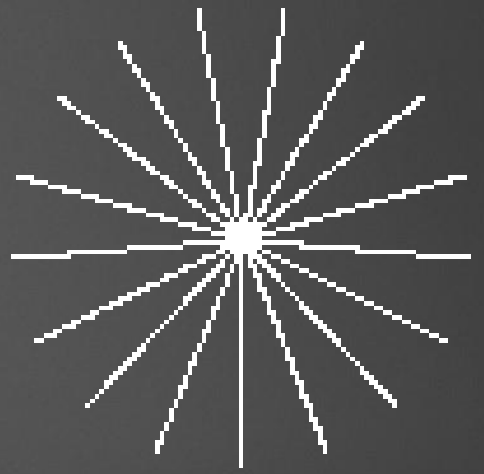
Простейшая прямая.

```
void line(int x0, int y0, int x1, int y1,
Image &image, Color color) {
    for (float t=0.0; t<1.0; t+=0.01) {
        int x = x0*(1.-t) + x1*t;
        int y = y0*(1.-t) + y1*t;
        image.SetPixel(x, y, color);
    }
}
```

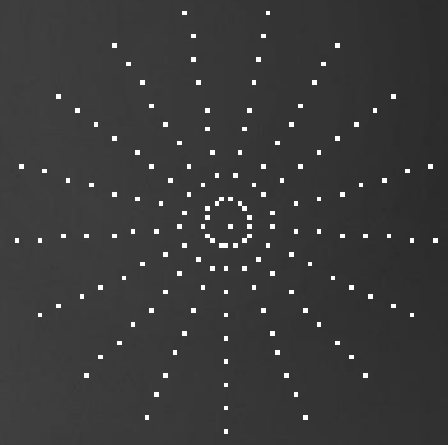
- Отрисовка пикселя с координатами (x, y) с шагом t.
- Шаг t определяет количество точек между началом и концом прямой.

Простейшая прямая. Проблемы.

```
void line(int x0, int y0, int x1, int y1,
Image &image, Color color) {
    for (float t=0.0; t<1.0; t+=0.01) {
        int x = x0*(1.-t) + x1*t;
        int y = y0*(1.-t) + y1*t;
        image.SetPixel(x, y, color);
    }
}
```



$\Delta t = 0.01$



$\Delta t = 0.1$

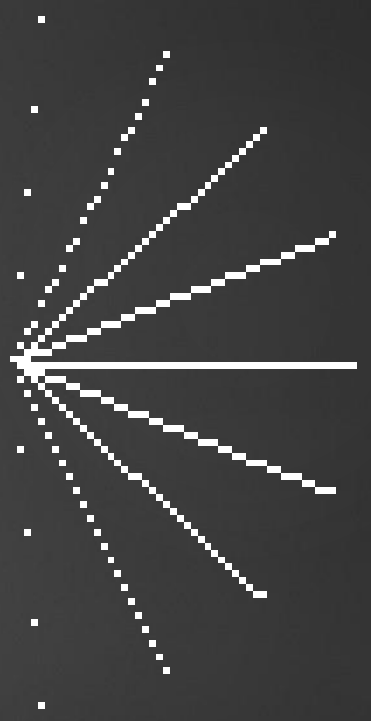
Второй вариант.

```
void line(int x0, int y0, int x1, int y1,
Image &image, Color color) {
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        image.SetPixel(x, y, color);
    }
}
```

- Отрисовка пиксела для каждого значения X.
- Значение Y рассчитывается на основе значения X.

Второй вариант. Всё ещё проблемы.

```
void line(int x0, int y0, int x1, int y1,
Image &image, Color color) {
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        image.SetPixel(x, y, color);
    }
}
```



$\Delta t = 0.01$

$\Delta t = 0.1$

Вносим правки. Наклон прямой.

```
bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
        std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }

<…>

if (steep) {
            image.SetPixel(y, x, color);
        } else {
            image.SetPixel(x, y, color);
        }
```

- Если смещение по оси X меньше смещения по оси Y, меняем местами координаты.
- На шаге отрисовки, если координаты менялись местами используем X вместо Y, и наоборот.

Вносим правки. Направление прямой.

```
if (x0>x1) {
      swap(x0, x1);
      swap(y0, y1);
   }
```

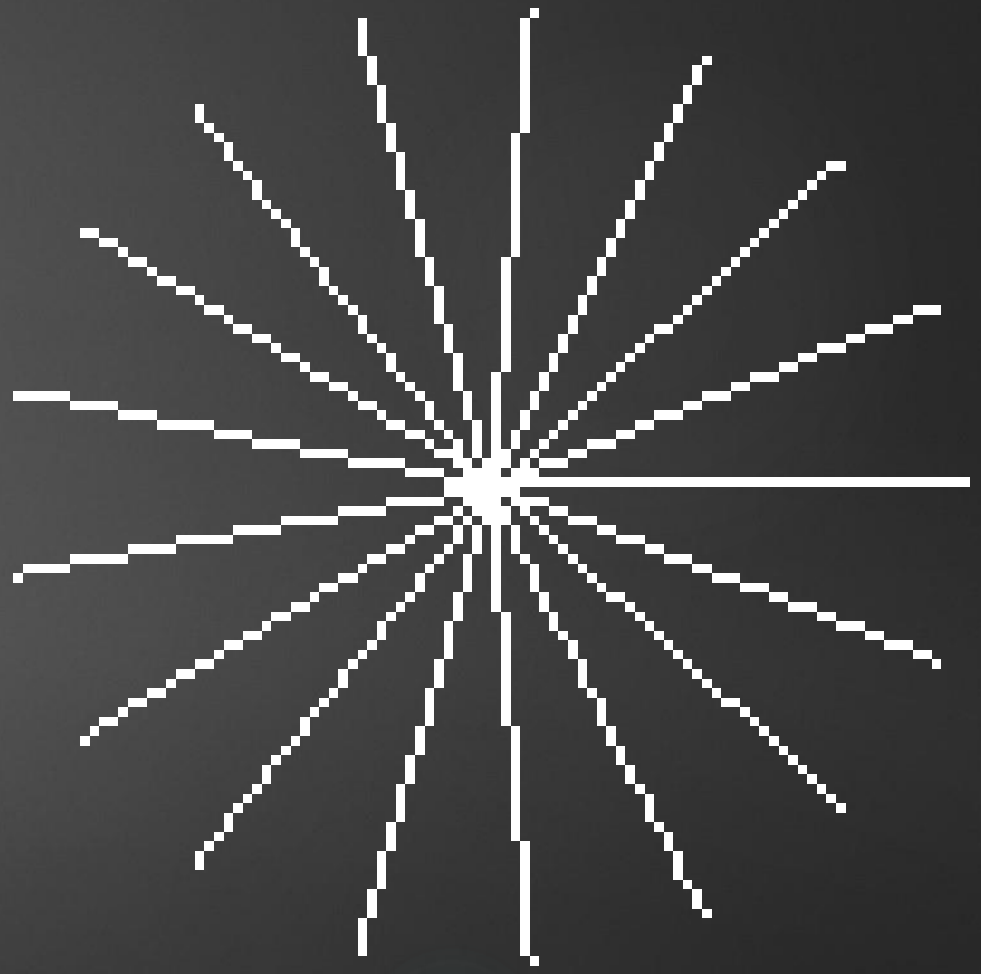
- Если начало прямой находится правее конца, меняем начало и конец местами.

Третий вариант.

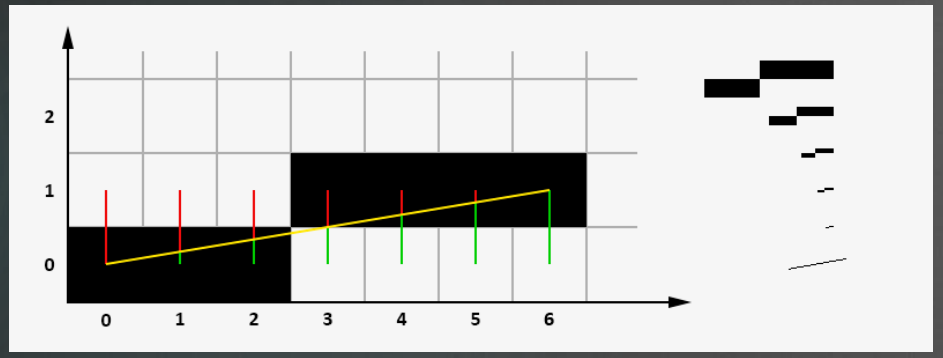
```
void line(int x0, int y0, int x1, int y1, Image &image, Color
color) {
    bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
        std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }
    if (x0>x1) { // make it left-to-right
        std::swap(x0, x1);
        std::swap(y0, y1);
    }
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        if (steep) {
            image.SetPixel(y, x, color);
        } else {
            image.SetPixel(x, y, color);
        }
    }
}
```

Третий вариант (он же первый рабочий)

```
void line(int x0, int y0, int x1, int y1, Image &image, Color
color) {
    bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
        std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }
    if (x0>x1) { // make it left-to-right
        std::swap(x0, x1);
        std::swap(y0, y1);
    }
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        if (steep) {
            image.SetPixel(y, x, color);
        } else {
            image.SetPixel(x, y, color);
        }
    }
}
```



Алгоритм Брезенхема.

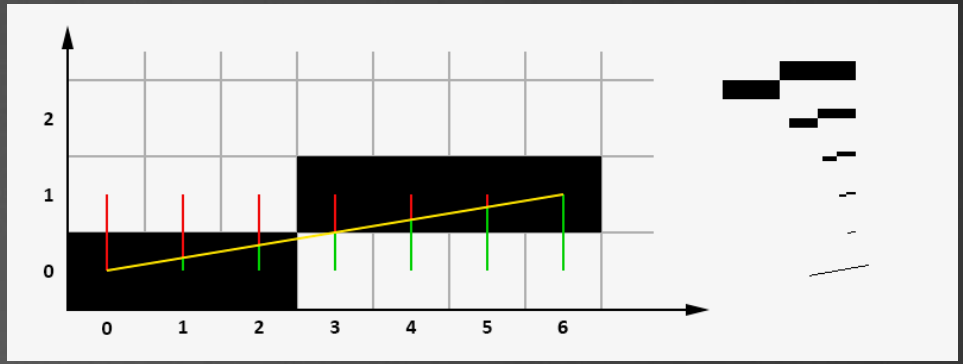


- Идея алгоритма Брезенхема в том, чтобы на каждом шаге вычислять смещение по оси Y относительно центра пиксела и при превышении значения 0,5 сдвигать отображаемый пиксель на одно положение вверх/вниз.

Четвёртый вариант. «(Почти) Идеальный»

```
void line(int x0, int y0, int x1, int y1, Image &image, Color
color) {
    int dx = x1-x0;
    int dy = y1-y0;
    float derror = std::abs(dy/float(dx));
    float error = 0;
    int y = y0;

    for (int x=x0; x<=x1; x++) {
        image.SetPixel(x, y, color);
        error += derror;
        if (error>.5) {
            y += (y1>y0?1:-1);
            error -= 1.;
        }
    }
}
```

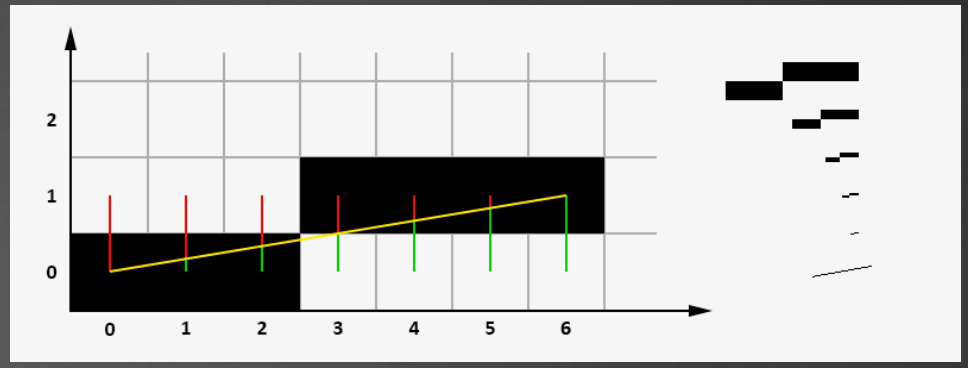


- Для упрощения восприятия предполагаем, что смещение по X больше, чем по Y, а начало отрезка расположено левее конца.
- derror – отношение сдвига по Y и сдвига по X. Фактически – значение, добавляемое на каждом шаге к смещению по Y.
- error – переменная, накапливающая смещение по Y.
- Как только error превышает 0,5 – увеличиваем/уменьшаем значение y на 1.

Четвёртый вариант. «(Почти) Идеальный» (окончательная версия)

```
void line(int x0, int y0, int x1, int y1, Image &image, Color color) {
    bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
        std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }
    if (x0>x1) { // make it left-to-right
        std::swap(x0, x1);
        std::swap(y0, y1);
    }
    int dx = x1-x0;
    int dy = y1-y0;
    float derror = std::abs(dy/float(dx));
    float error = 0;
    int y = y0;

    for (int x=x0; x<=x1; x++) {
        if (steep) {
            image.SetPixel(y, x, color);
        } else {
            image.SetPixel(x, y, color);
        }
        error += derror;
        if (error>.5) {
            y += (y1>y0?1:-1);
            error -= 1.;
        }
    }
}
```



- Добавляем все «улучшения» предыдущих этапов

Задача на лабораторную

«Проволочный» рендер

- Загрузить модель в формате obj из файла.
- Нарисовать все рёбра модели без учёта перспективы, глубины, невидимых границ и т.д.