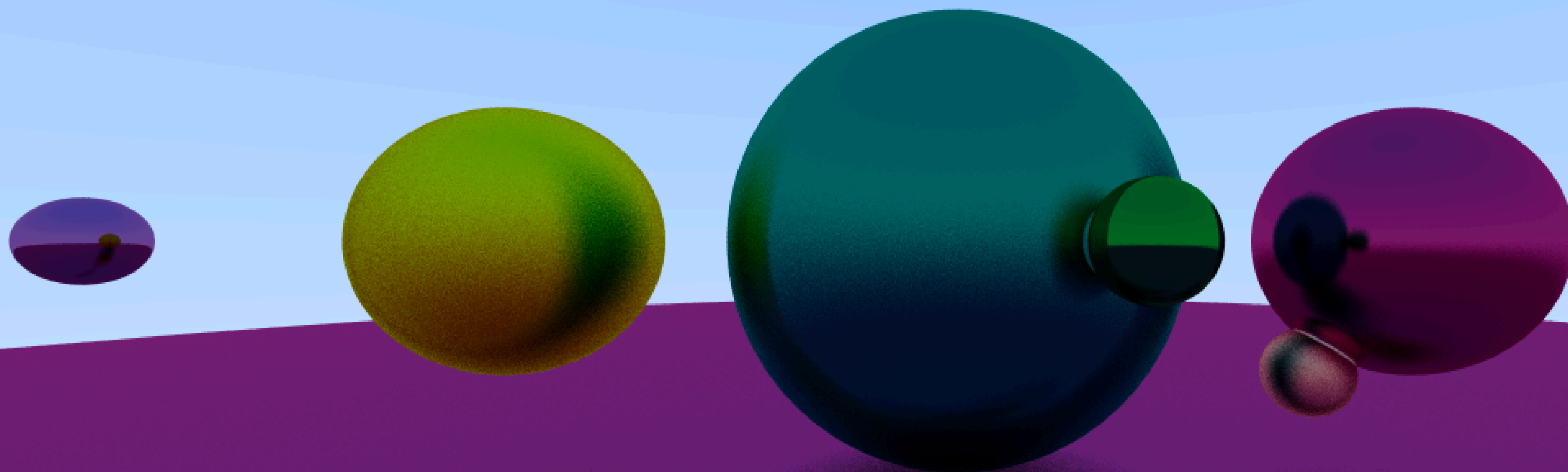
Алексей Лавренюк @direvius



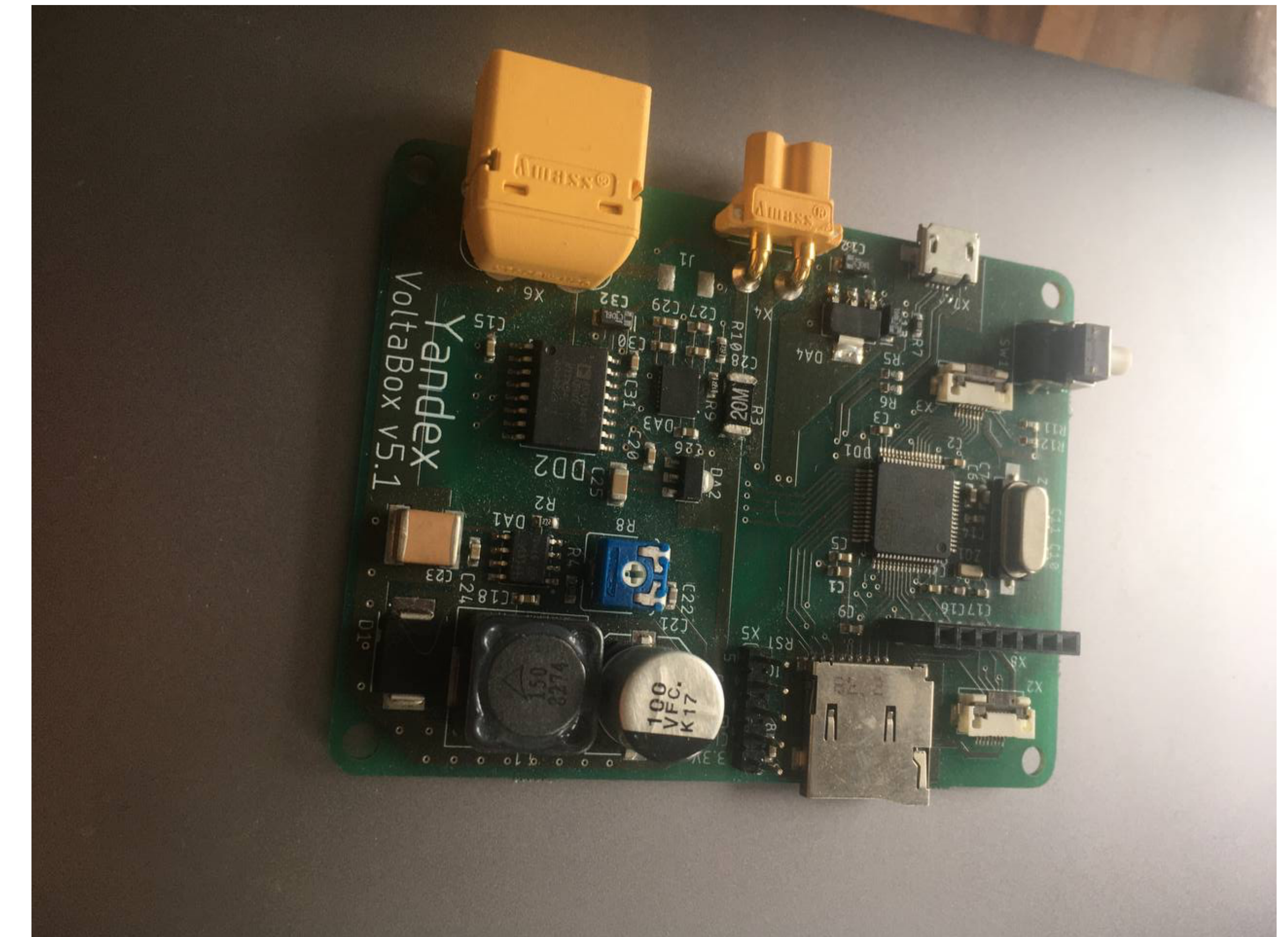
Raytracer in Python

можно ли догнать Rust?

О себе

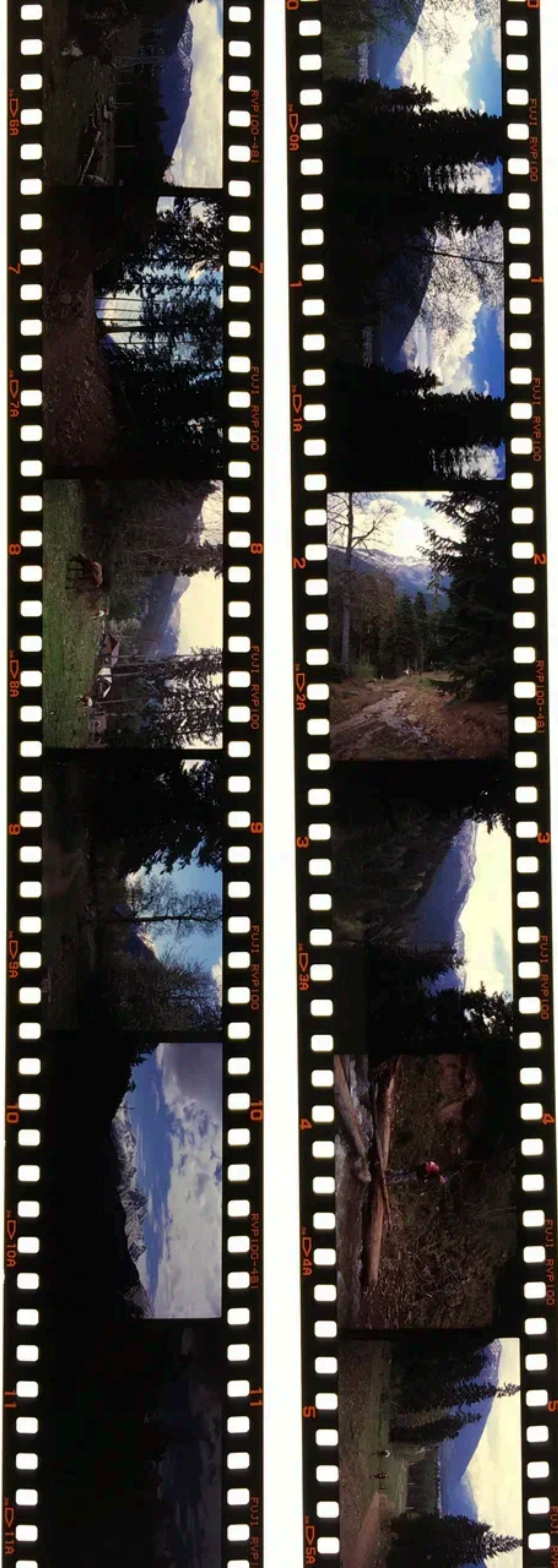
мои проекты

- Яндекс.Танк и Pandora доклад Саши Иванова: clck.ru/36ZVEw
- VoltaBox clck.ru/36ZVBN
- Гиперкуб clck.ru/36ZVCv
- Яндекс Ровер clck.ru/34gQ7V



О себе

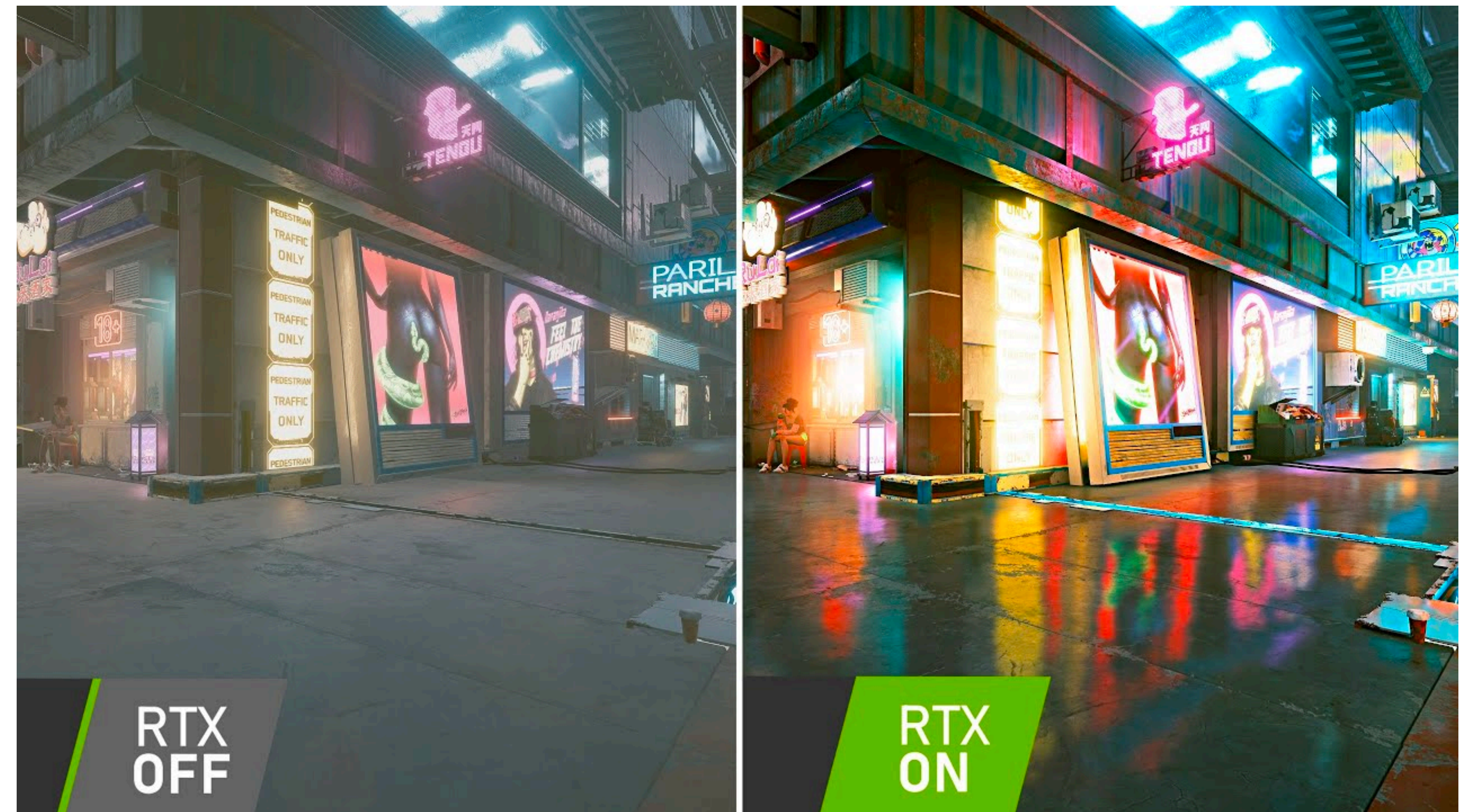
мои увлечения



О чем доклад

и почему я об этом говорю

- just for fun, разбираем "машинку"
- рейтрейсинг теперь везде
- сначала алгоритм трассировки, потом попробуем догнать Rust



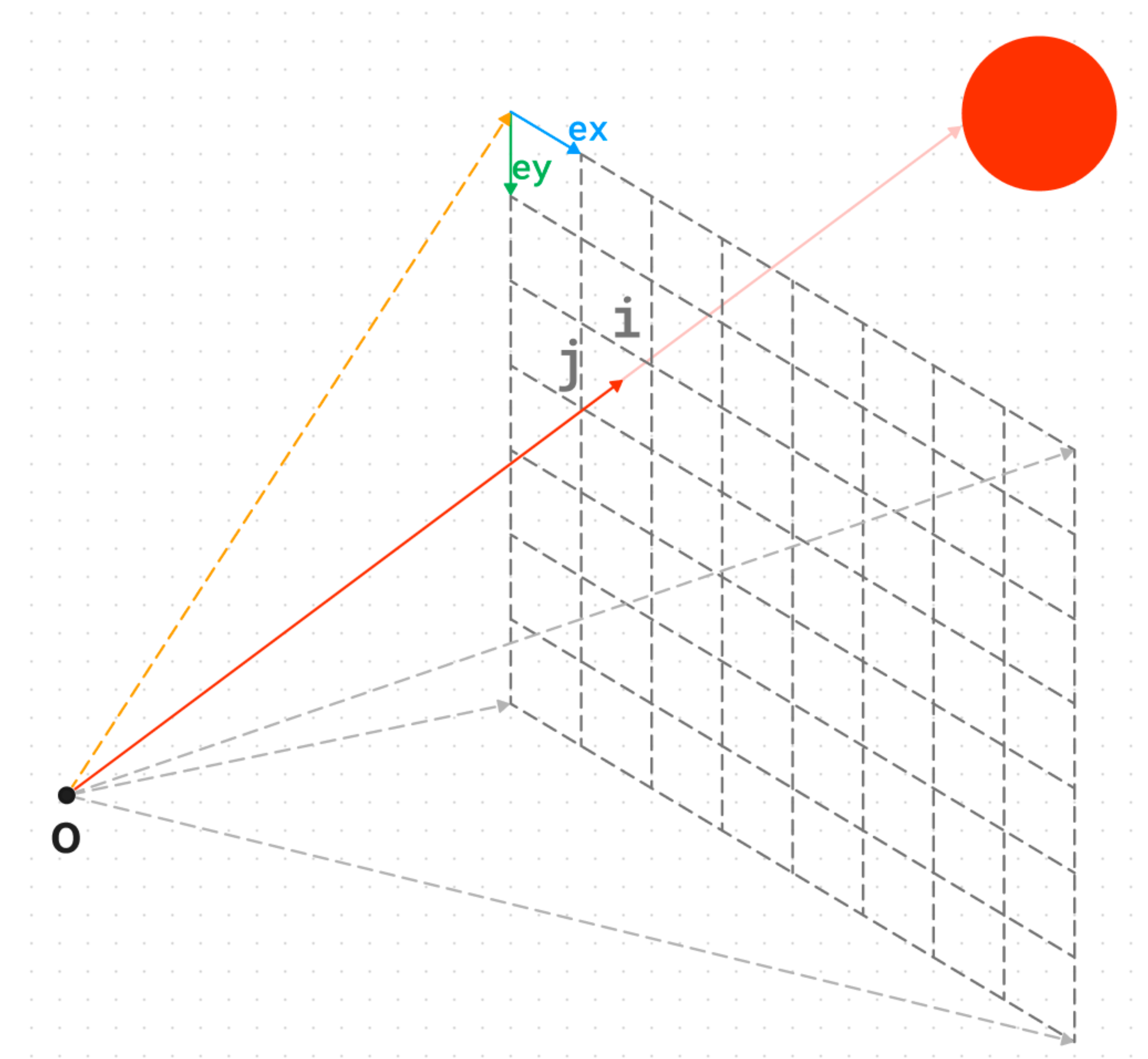
Алгоритм

Альбрехт Дюрер 1532

Камера

какого цвета этот пиксель?

- луч — это точка (начала) и вектор (направления)
- сгенерировать лучи для каждого пикселя



Лучи

```
1.def get_ray(self, i: int, j: int) -> Ray:
2.    return Ray(
3.        self.origin,
4.        self.upper_left + self.ex * i - self.ey * j,
5.    )
6.
```

Окружение

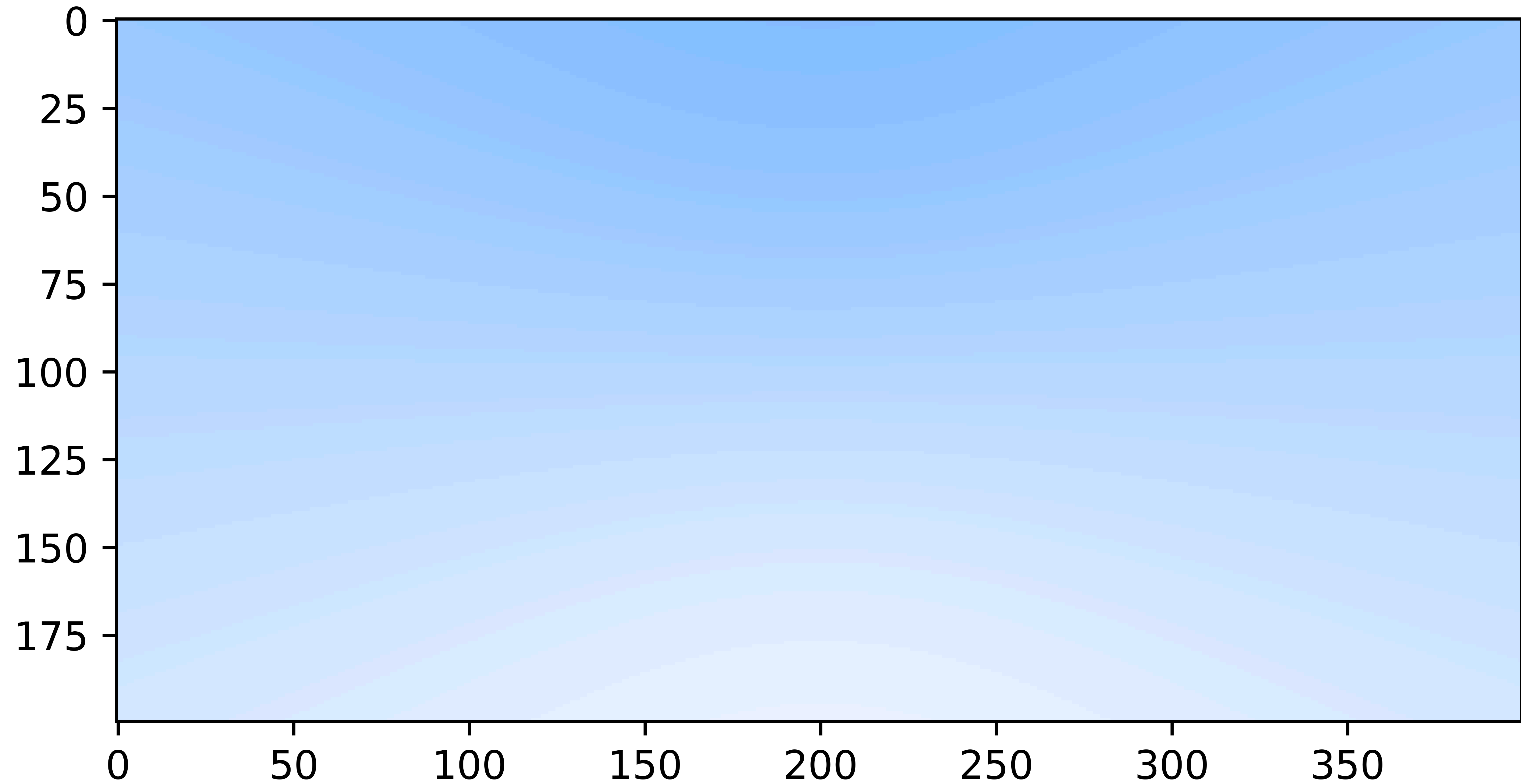
какого цвета этот пиксель?

- если не попали в объект,
  попали в небо
- можно нарисовать градиент
- можно сложнее —
  использовать карту окружения

Небо

простой градиент по вертикали

```
1.def env_color(r: Ray) -> Vector3:
2.    t = (r.direction.y + 1) * 0.5
3.    env_color = (1 - t) * Vector3(1, 1, 1) + t * Vector3(0.5, 0.7, 1)
4.    return env_color
5.
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Что будем рисовать

натюрморт из сфер

```
1.SceneBuilder()
2.      .add_sphere(0, 0, -1.5, 0.5)  # central
3.      .add_sphere(1, 0, -1.5, 0.3)  # right
4.      .add_sphere(0.3, 0, -1, 0.1)
5.      .add_sphere(0.8, -0.3, -1.5, 0.1)
6.      .add_sphere(-1, 0, -1.5, 0.3)  # left
7.      .add_sphere(-2, 0, -1.5, 0.1)
8.      .add_sphere(0, -100.5, -1.5, 100)  # The Earth
9.
```

Сфера — это центр и радиус (и материал, в нашем случае, случайный)

Как найти объект

черная сфера в темной комнате

Допустим, что луч пересекается со сферой в точке p при параметре, равном t:

$$\vec{p} = \vec{o} + \vec{d}\,t$$

$$|\,\vec{p} - \vec{c}\,| = r$$

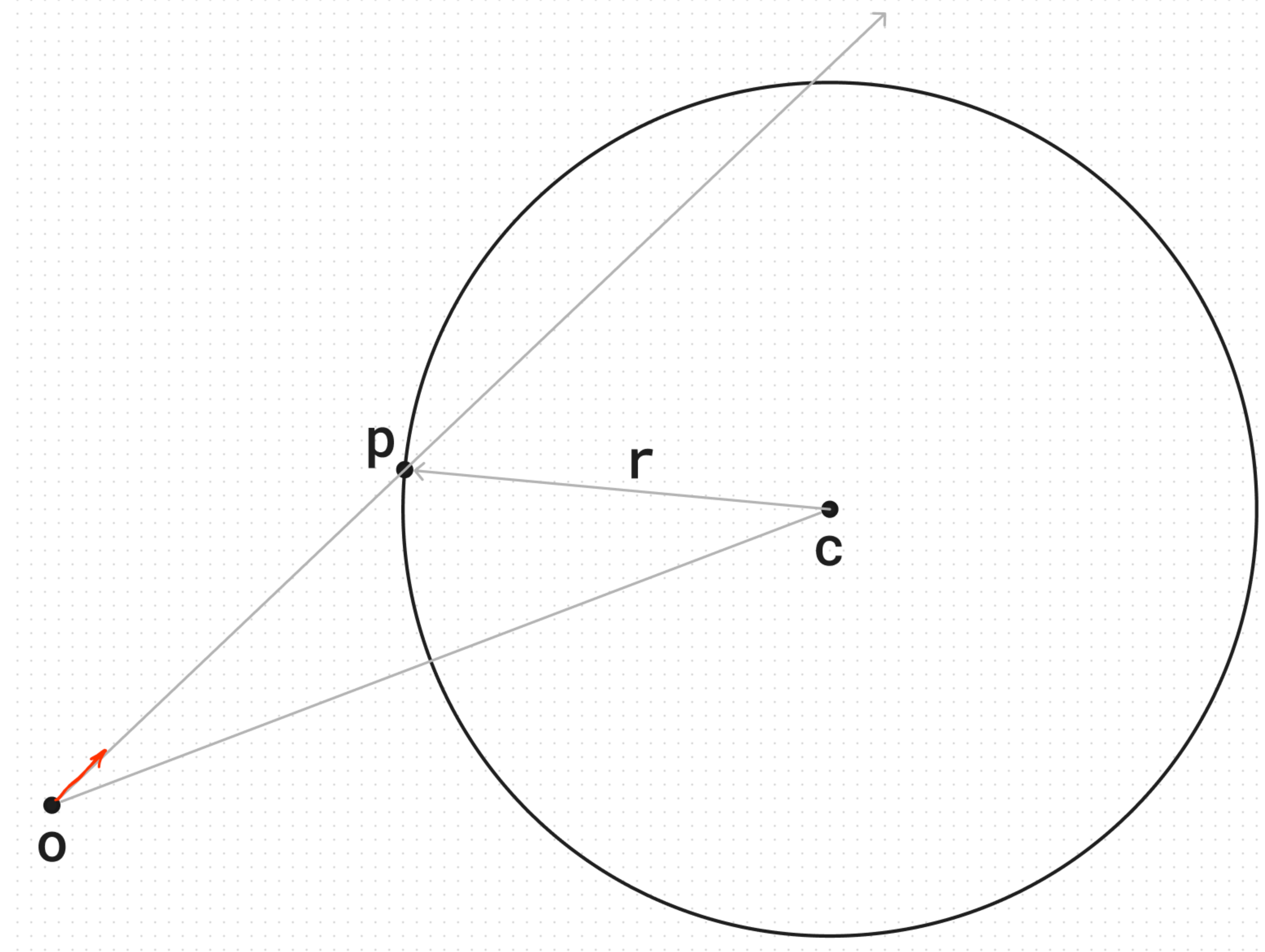
$$(\vec{o} + \vec{d}\,t - \vec{c})^2 = r^2; \vec{l} = \vec{o} - \vec{c}$$

$$\vec{d}^{\,2}t^2 + 2\,\vec{d}\,\vec{l}\,t + \vec{l}^{\,2} - r^2 = 0$$

1

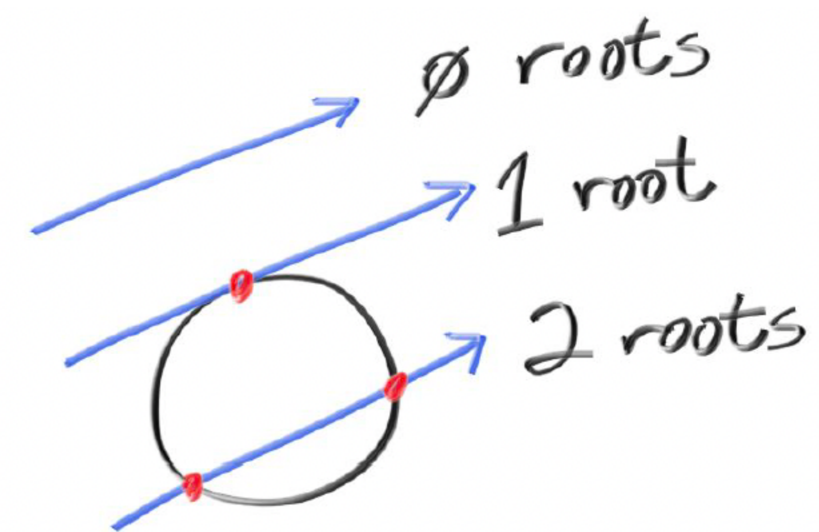
b

c



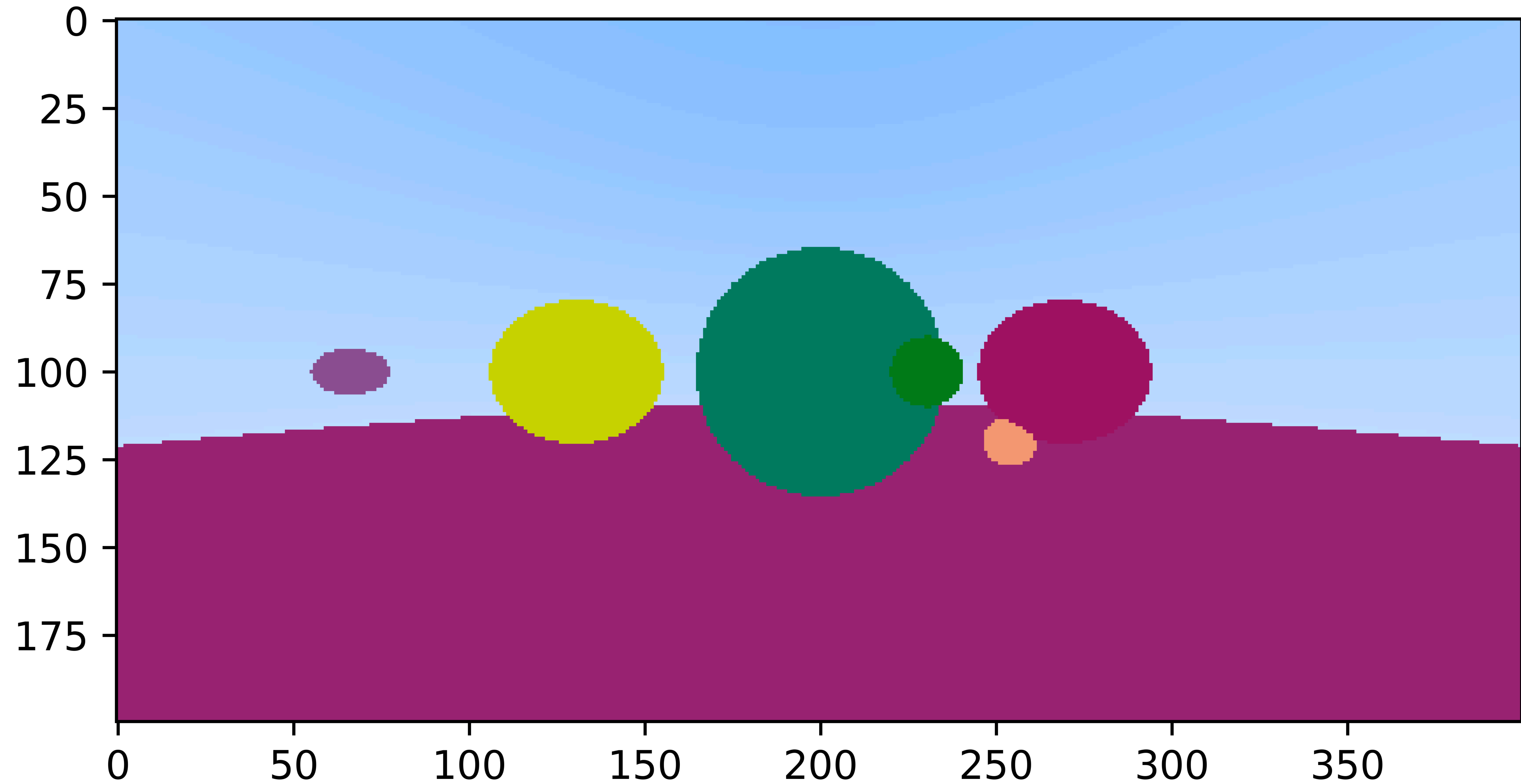
Пересечения

```
1.def hit(self, r: Ray) -> HitResult | None:
2.    l = r.origin - self.center
3.    b = l.dot(r.direction) * 2.0
4.    c = l.dot(l) - self.radius**2
5.    d = b * b - c * 4.0
6.    if d >= 0.0:
7.        ...
8.
```



Выбрать ближайшее пересечение

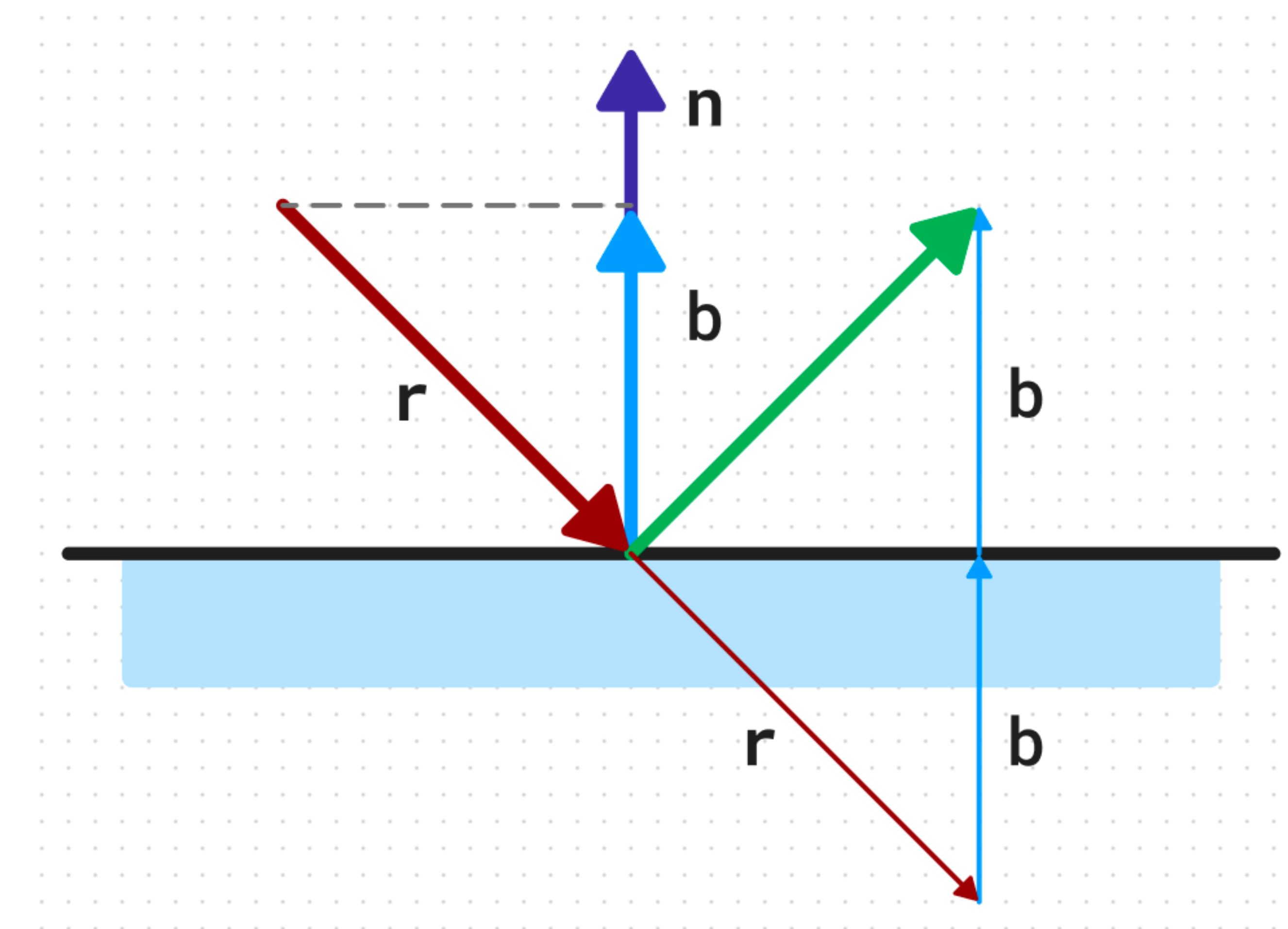
```
1.def hit(self, r: Ray) -> HitResult | None:
2.    hits = [hr for g in self.geometry if (hr := g.hit(r))]
3.    if len(hits) > 0:
4.        return min(hits, key=lambda hr: hr.t)
5.    else:
6.        return None
7.
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Идеальное отражение

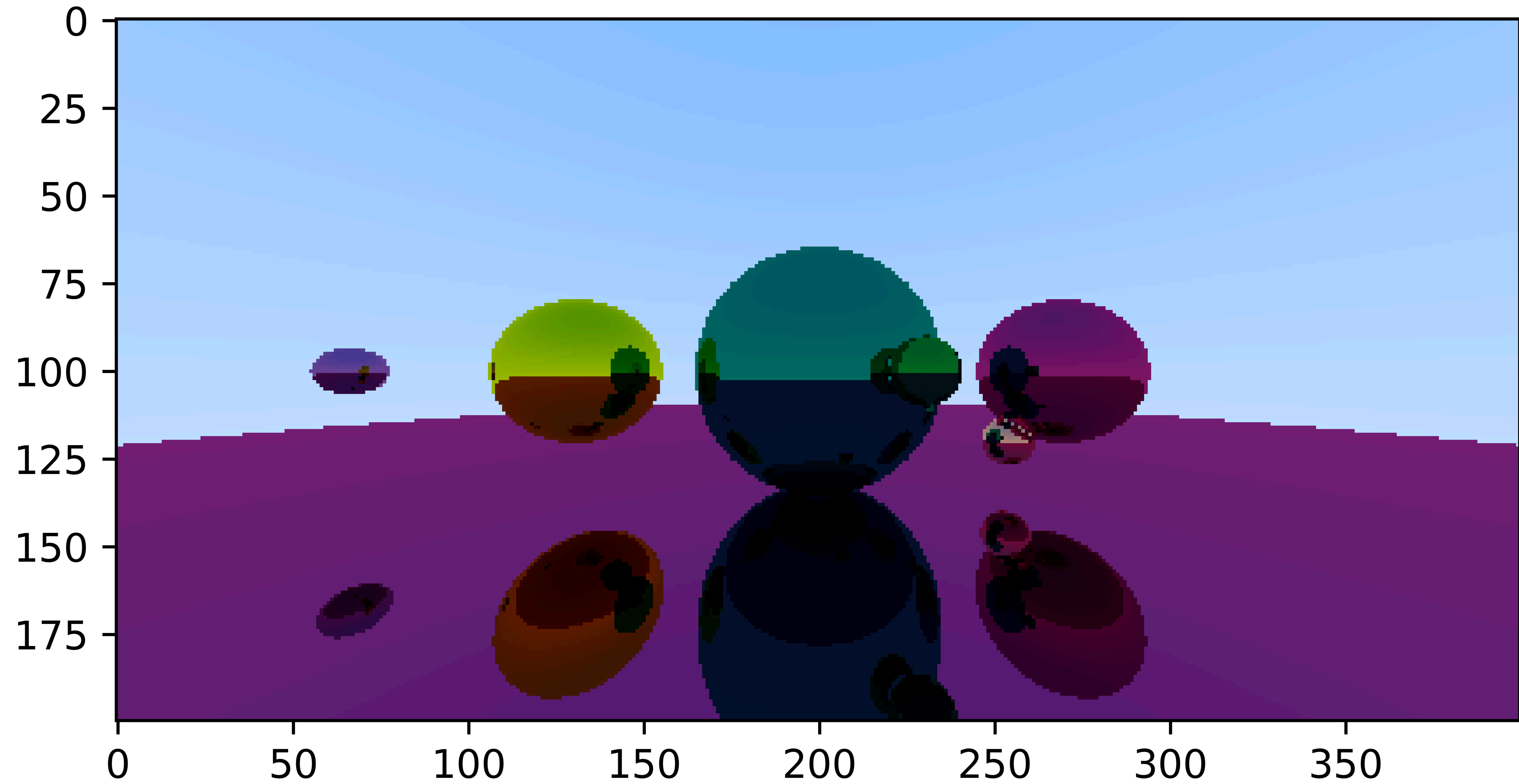
глянцевые поверхности

- угол падения равен углу отражения
- перемещаем "наблюдателя" в точку пересечения и рекурсивно наблюдаем цвет отраженного луча



Идеальное отражение

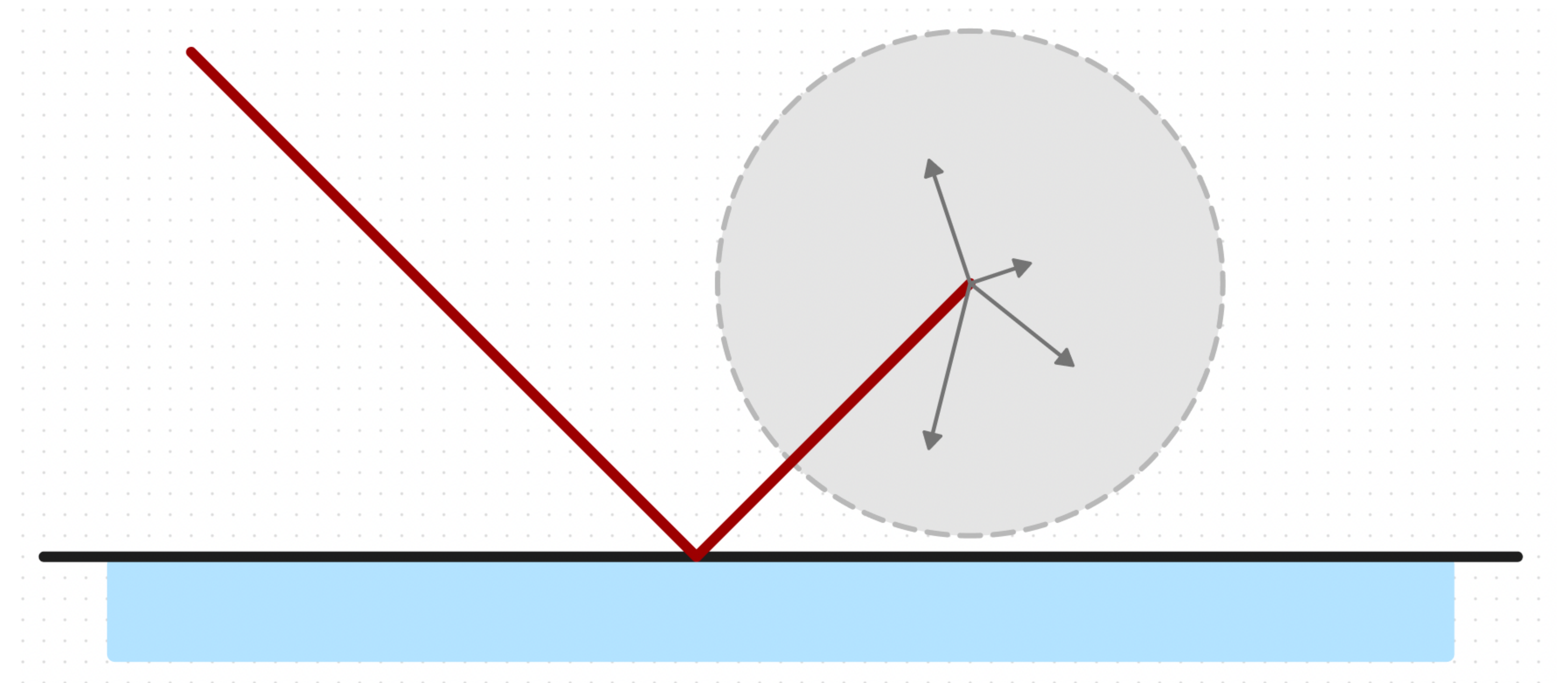
```
1.def reflect_direction(r: Vector3, n: Vector3) -> Vector3:
2.    return r - 2 * n * r.dot(n)
3.
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Матовое отражение

или великий рандом

- добавить произвольный вектор к отраженному лучу

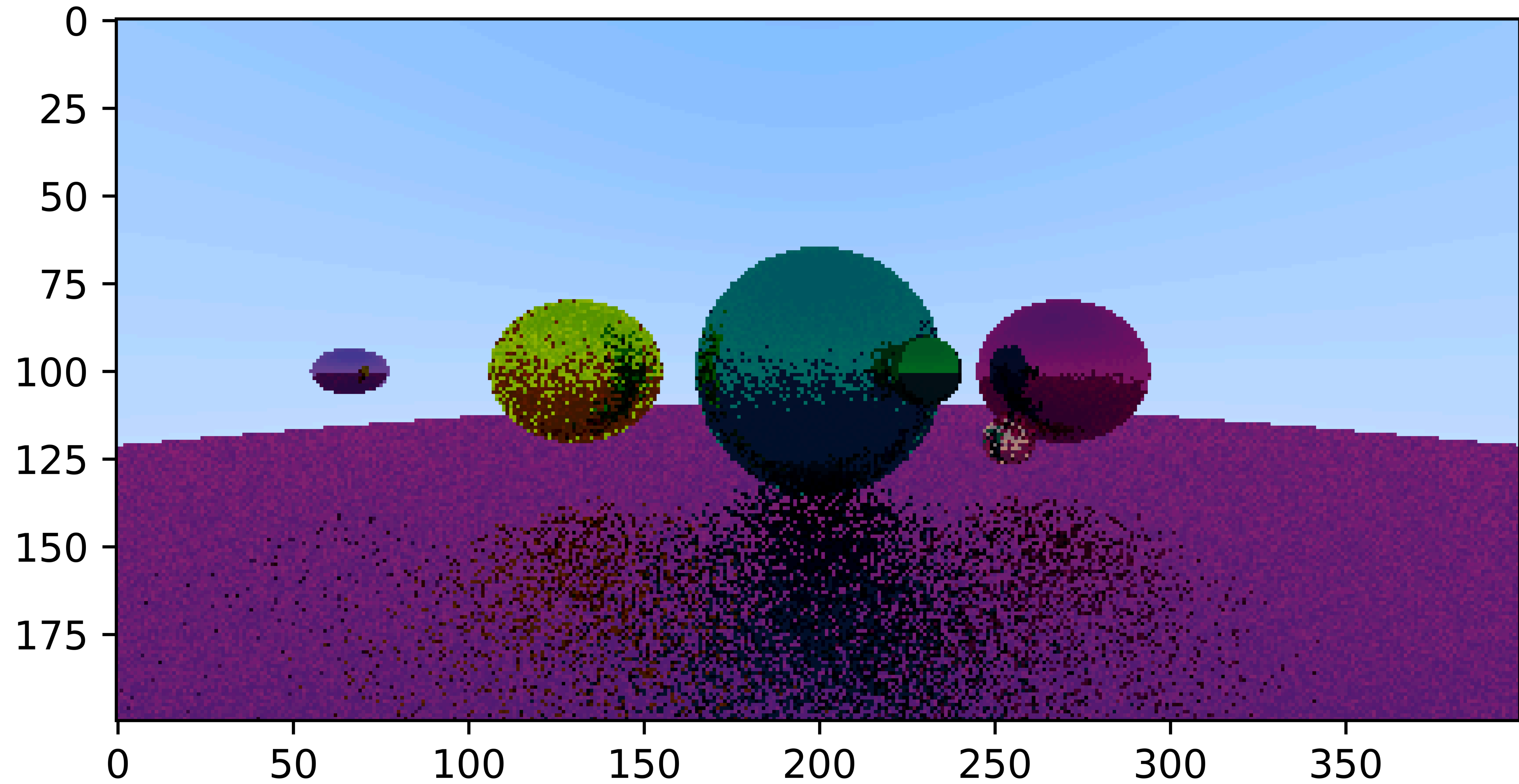


Произвольный вектор

```
1.def random_in_unit_sphere() -> Vector3:
2.    while True:
3.        v = 2 * Vector3(random(), random(), random()) - Vector3(1, 1, 1)
4.        if abs(v) < 1:
5.            return v
6.
```

Комбинированное отражение

```
1.def collide(self, point: Vector3, normal: Vector3, r: Ray) -> CollideResult:
2.    collided = Ray(
3.        origin=point,
4.        direction=r.direction.reflect_direction(normal) + Vector3.random_in_unit_sphere() * self.fuzz)
5.    return CollideResult(collided, self.attenuation)
6.
```

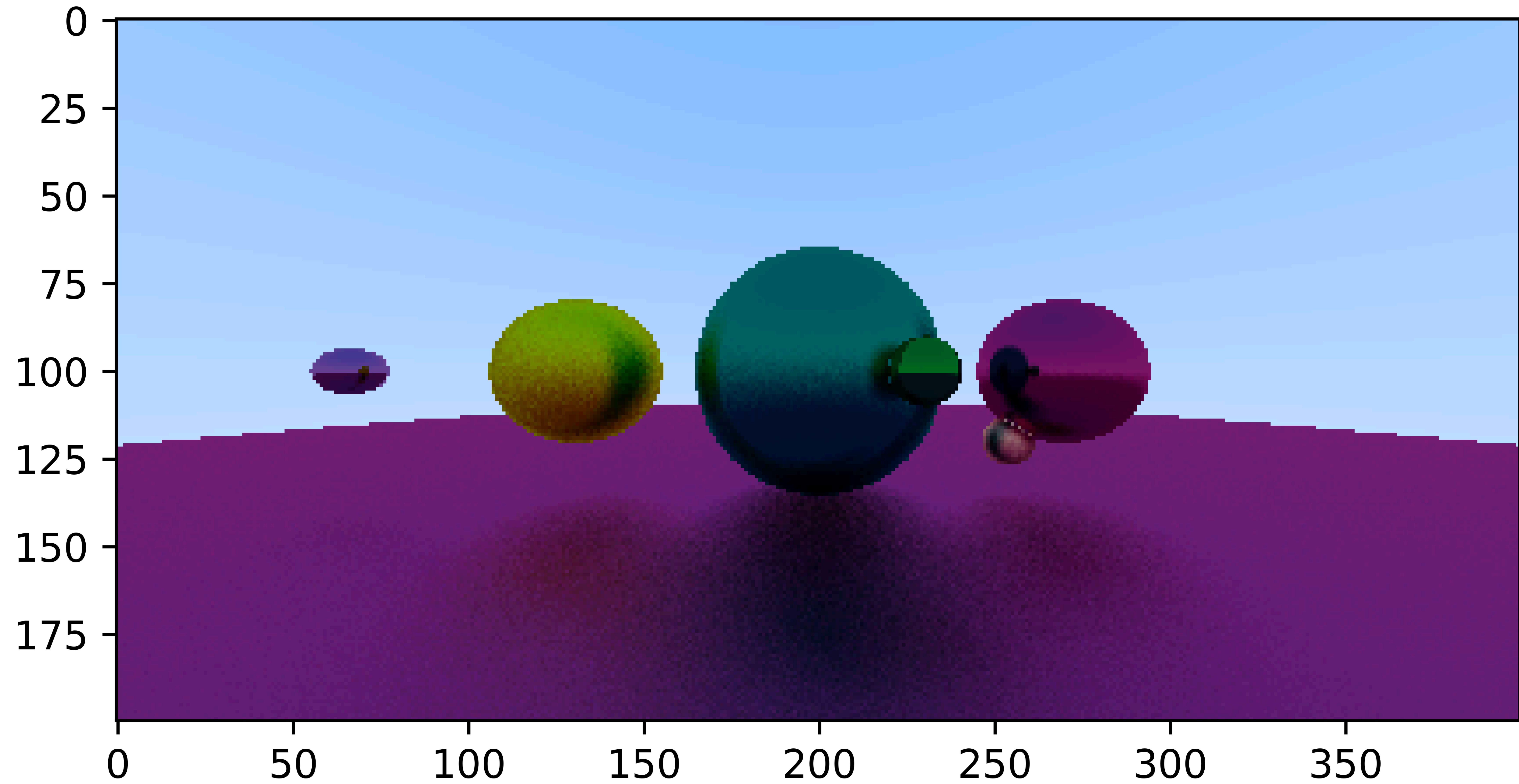


0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Несколько проходов

для усреднения случайных отражений

```
1.for _ in range(self.jitter_passes):
2.    r = self.get_ray(ix, iy)
3.    color += self.trace(r)
4.img[iy][ix] = color / self.jitter_passes
5.
```


0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Комбинированное отражение

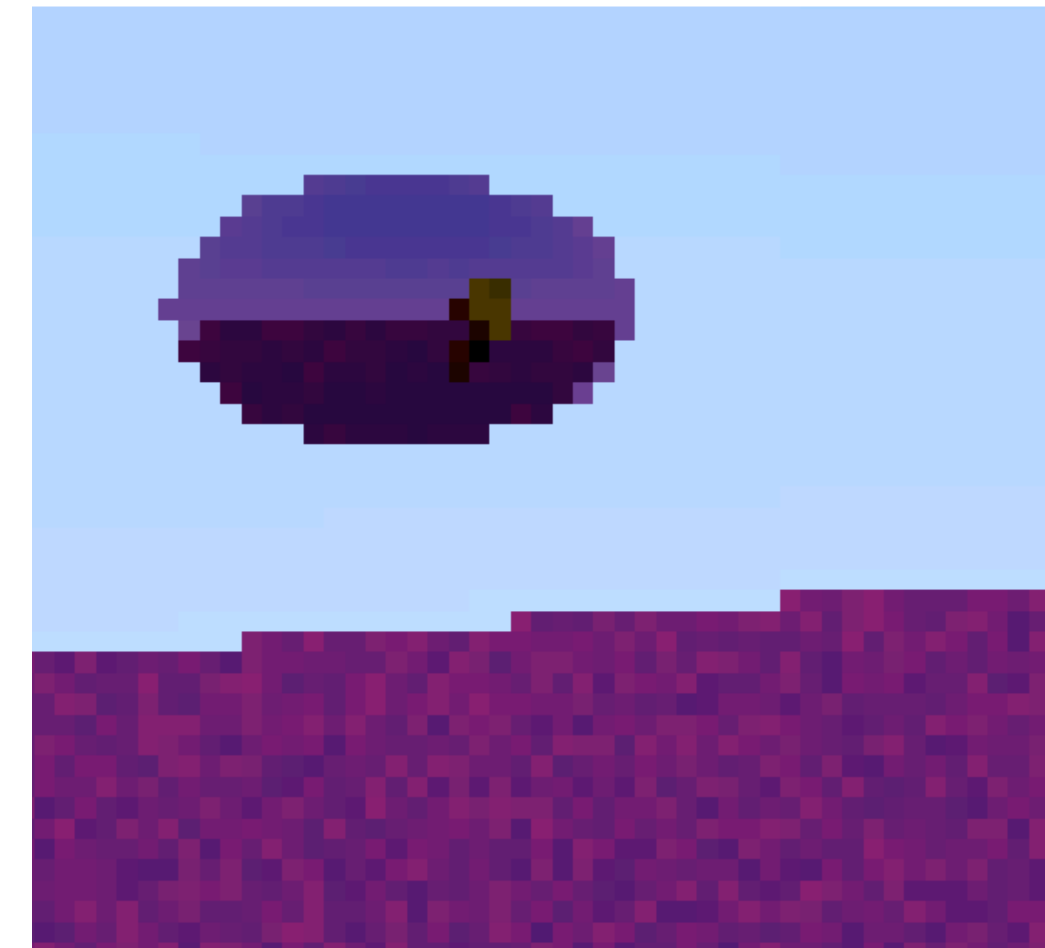
тут есть баг

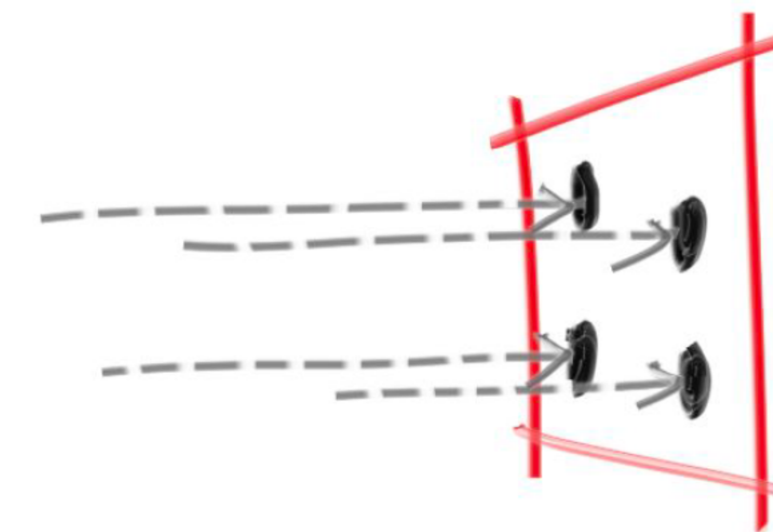
```
1.def collide(self, point: Vector3, normal: Vector3, r: Ray) -> CollideResult:
2.    collided = Ray(
3.        origin=point,
4.        direction=r.direction.reflect_direction(normal) + Vector3.random_in_unit_sphere() * self.fuzz)
5.    return CollideResult(collided, self.attenuation)
6.
```

Anti-aliasing

пикселы-окошки

- усреднить цвет нескольких лучей внутри одного пикселя



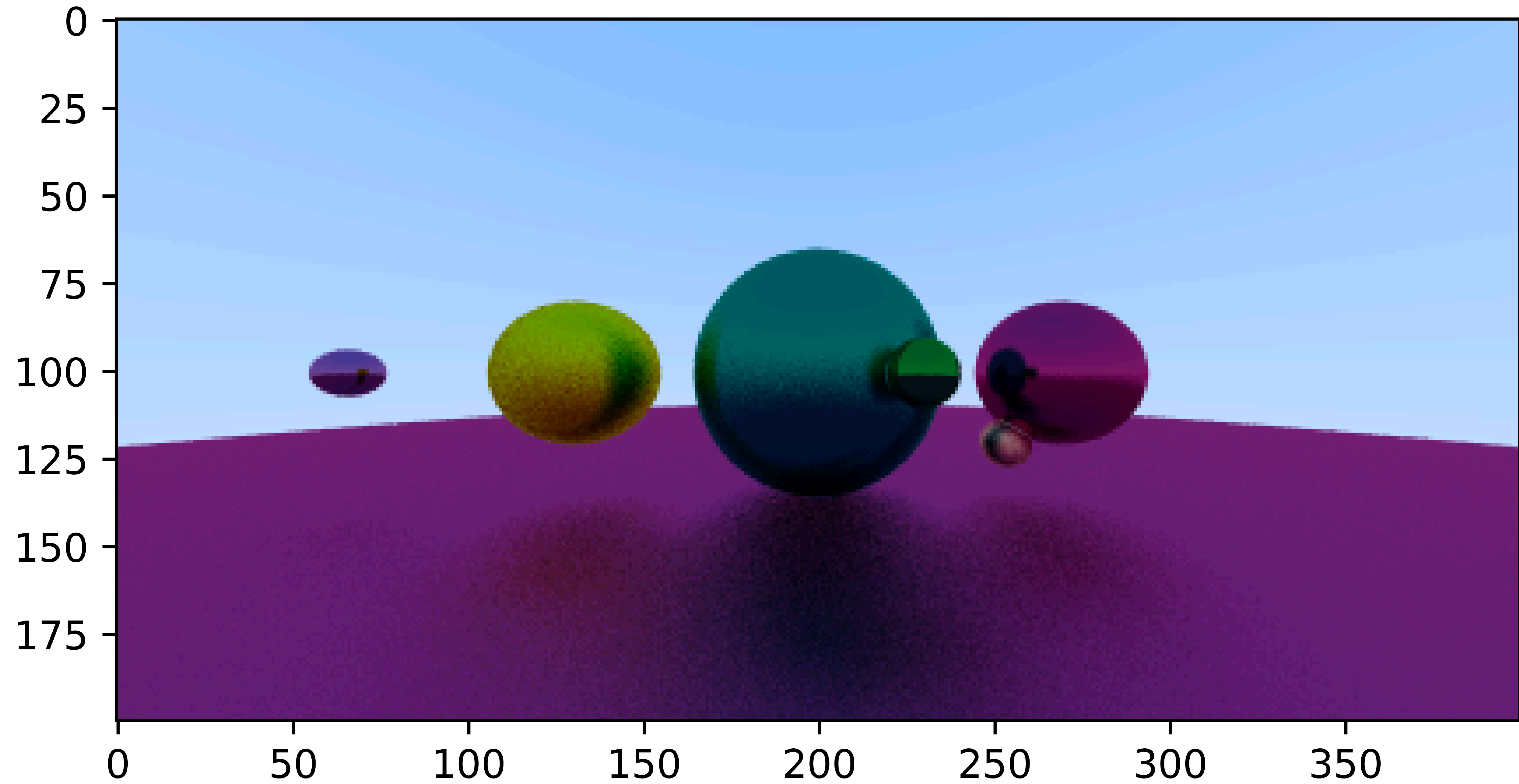


Anti-aliasing off

```
1.def get_ray(self, i: int, j: int) -> Ray:
2.    return Ray(
3.        self.origin,
4.        self.upper_left + self.ex * i - self.ey * j,
5.    )
6.
```

Anti-aliasing on

```
1.def get_ray(self, i: int, j: int) -> Ray:
2.    return Ray(
3.        self.origin,
4.        self.upper_left + self.ex * (i + random()) - self.ey * (j + random()),
5.    )
6.
```



0
25
50
75
100
125
150
175
0
50
100
150
200
250
300
350

Чистый Python

500 строк кода

```
1.@frozen
2.class Vector3:
3.
4.    x: float
5.    y: float
6.    z: float
7.
8.    def __add__(self, other):
9.        return Vector3(self.x + other.x, self.y + other.y, self.z + other.z)
10.
11.    def __sub__(self, other):
12.        return Vector3(self.x - other.x, self.y - other.y, self.z - other.z)
13.
14.    def __abs__(self):
15.        return math.sqrt(self.x**2 + self.y**2 + self.z**2)
16.
17.    ...
18.
19.
```

Сколько займет генерация картинки 400x200 в 64 прохода?

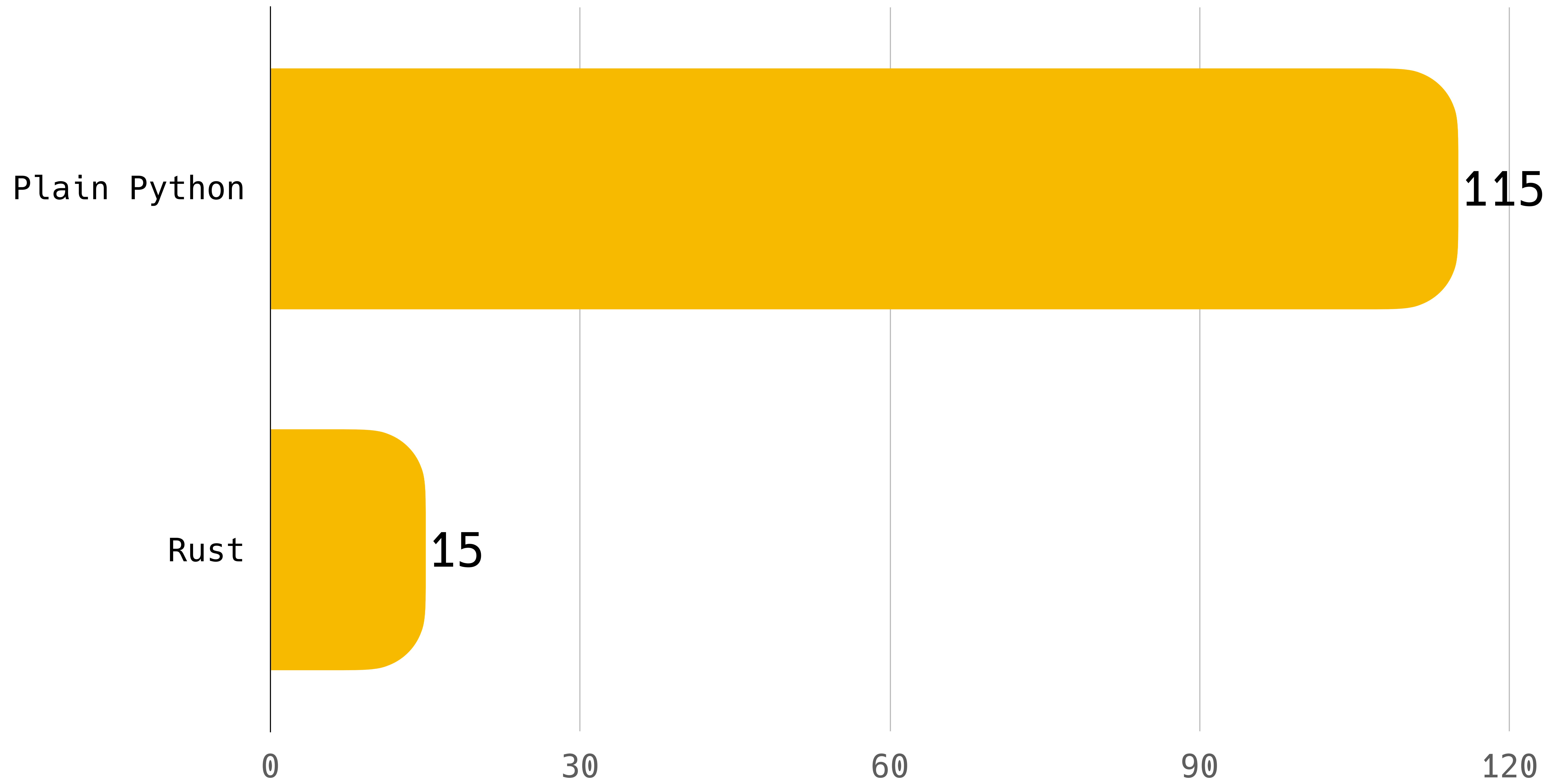
```
➜  rt git:(master) ✗ python ./main.py -a plain
2023-10-18 15:33:40.202 | INFO     | __main__:main:28 - Using plain implementation. Creating scene...
2023-10-18 15:33:40.202 | INFO     | __main__:main:40 - Tracing...
2023-10-18 15:35:35.191 | INFO     | __main__:main:44 - Tracing finished. It took 114.987 seconds
```

А то же самое на Rust?

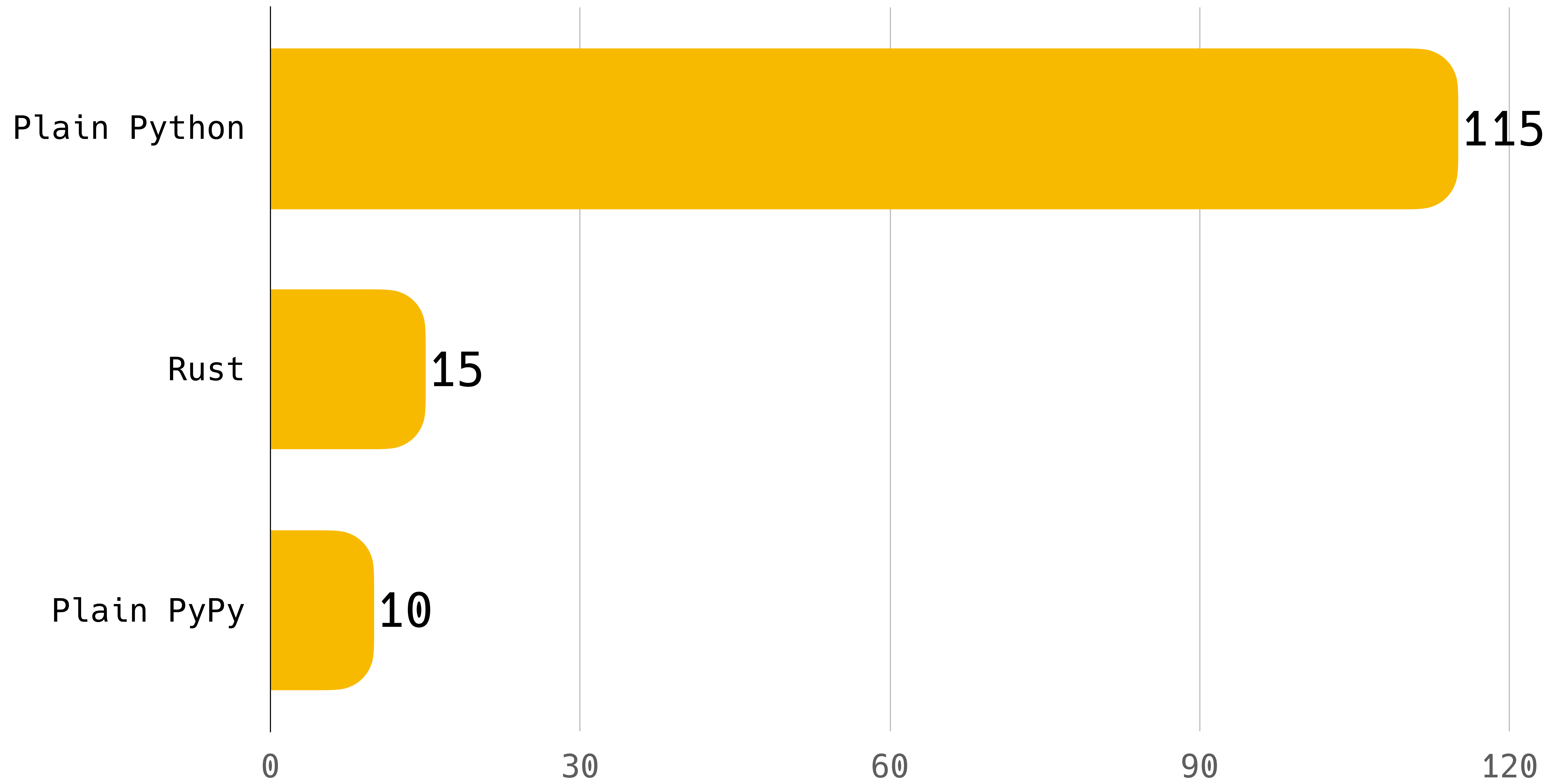
```
➜  rt git:(master) ✗ python ./main.py -a plain
2023-10-18 15:33:40.202 | INFO     | __main__:main:28 - Using plain implementation. Creating scene...
2023-10-18 15:33:40.202 | INFO     | __main__:main:40 - Tracing...
2023-10-18 15:35:35.191 | INFO     | __main__:main:44 - Tracing finished. It took 114.987 seconds
```

```
➜  rt git:(master) ✗ time ./target/debug/rt
./target/debug/rt  14.93s user 0.03s system 98% cpu 15.120 total
```

Гонка начинается: PyPy


Plain Python
115
Rust
15
0
30
60
90
120

```
➜  rt git:(master) ✗ pypy3 ./main.py -a plain
2023-10-18 16:06:25.012 | INFO     | __main__:main:28 - Using plain implementation. Creating scene...
2023-10-18 16:06:25.013 | INFO     | __main__:main:40 - Tracing...
2023-10-18 16:06:34.798 | INFO     | __main__:main:44 - Tracing finished. It took 9.785 seconds
```

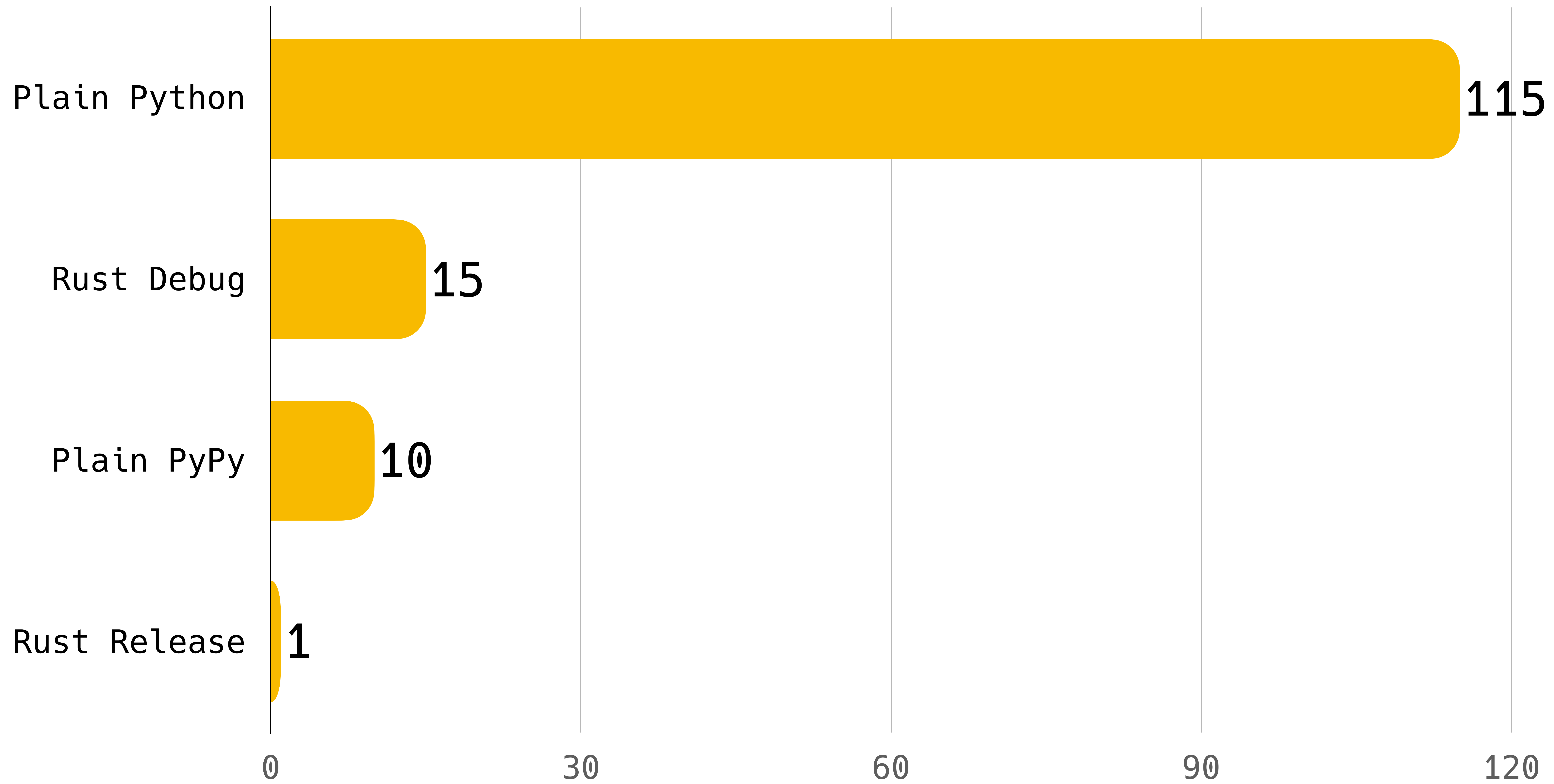


Plain Python
115
Rust
15
Plain PyPy
10
0
30
60
90
120

Спасибо за внимание!

```
➜  rt git:(master) ✗ time ./target/debug/rt
./target/debug/rt  14.93s user 0.03s system 98% cpu 15.120 total
```

```
➜  rt git:(master) ✗ time ./target/debug/rt
./target/debug/rt  14.93s user 0.03s system 98% cpu 15.120 total
```


Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
0
30
60
90
120

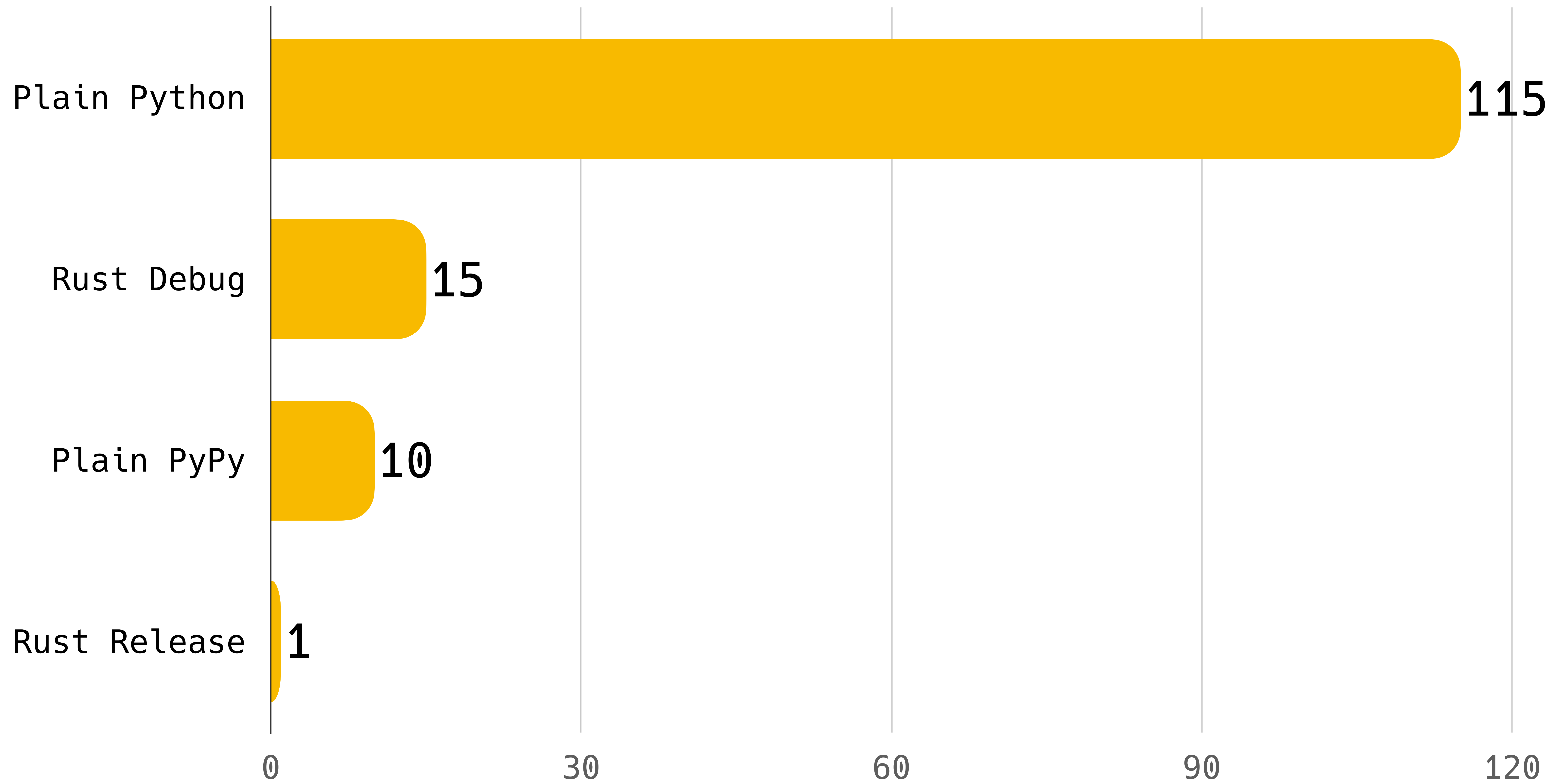
Профилировщик

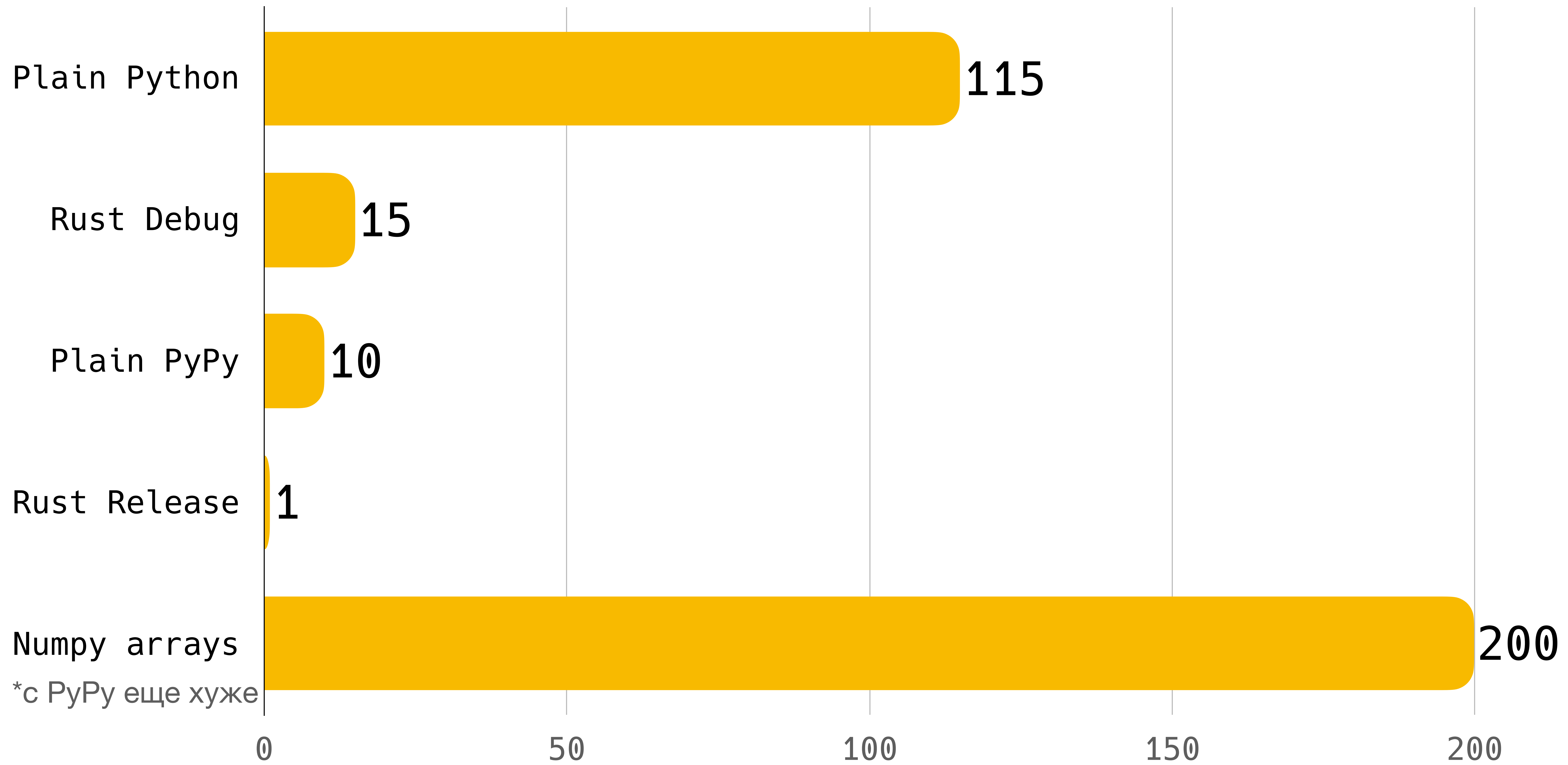
Plain Objects -> Numpy Array

Переназываем тип

и mypy подсказывает, что еще нужно сделать

```
Vector3 = Annotated[npt.NDArray[np.float64], Literal[3]]
```


Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
0
30
60
90
120


Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
Numpy arrays
200
*с PyPy еще хуже
0
50
100
150
200

Векторизация

Изменения

нужно больше, чем просто заменить функции

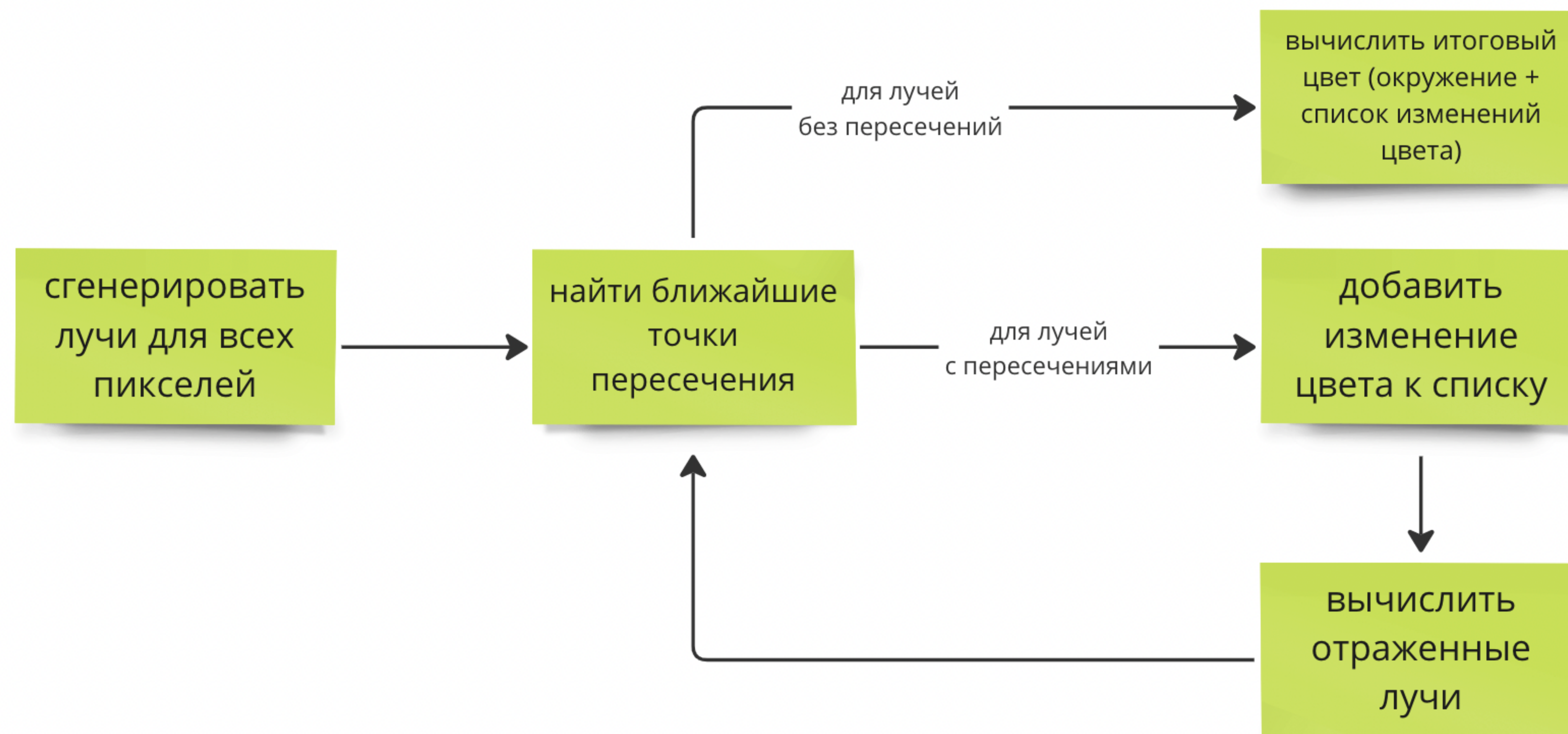
- все функции теперь принимают пачки векторов
- нет встроенной функции для скалярного произведения всех векторов из двух пачек, поэтому используем заклинание:

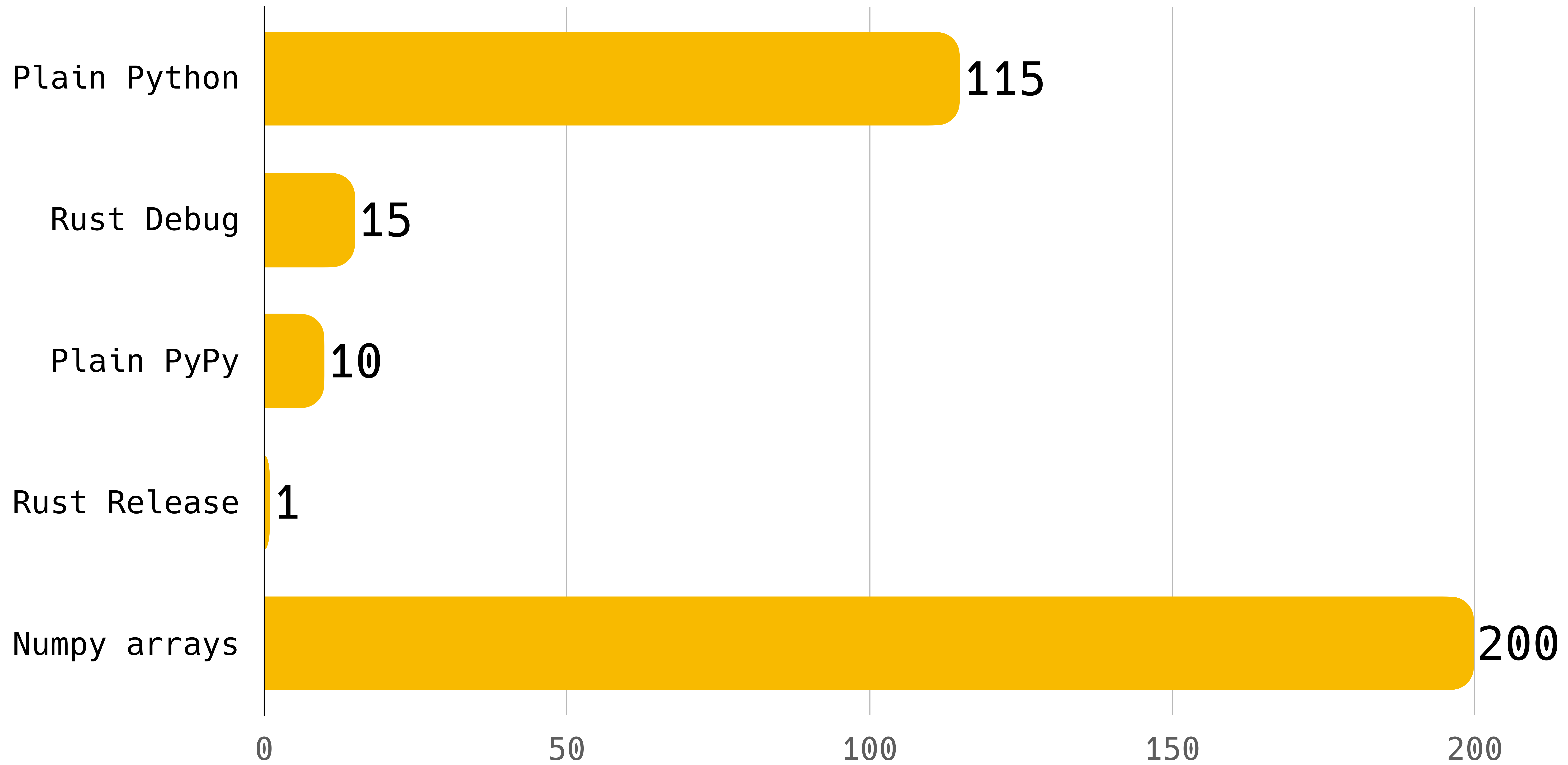
```
np.einsum("ij,ij->i", a, b)
```

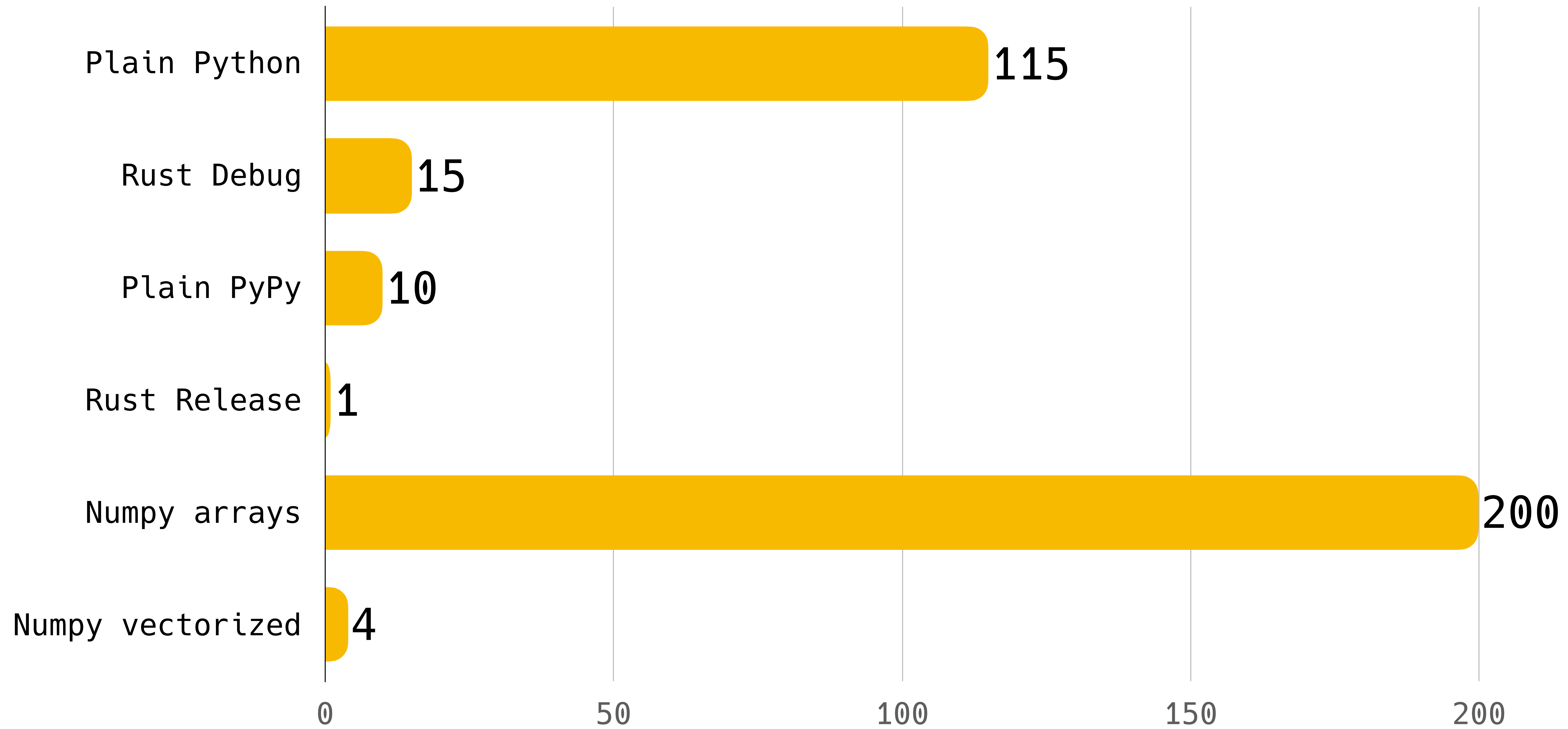
- нужно по-другому считать random in unit sphere, т.к. неудобно пересчитывать векторы с длиной больше 1 для пачки векторов

Пайплайн трассировки лучей

вместо взаимодействия объектов




Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
Numpy arrays
200
0
50
100
150
200



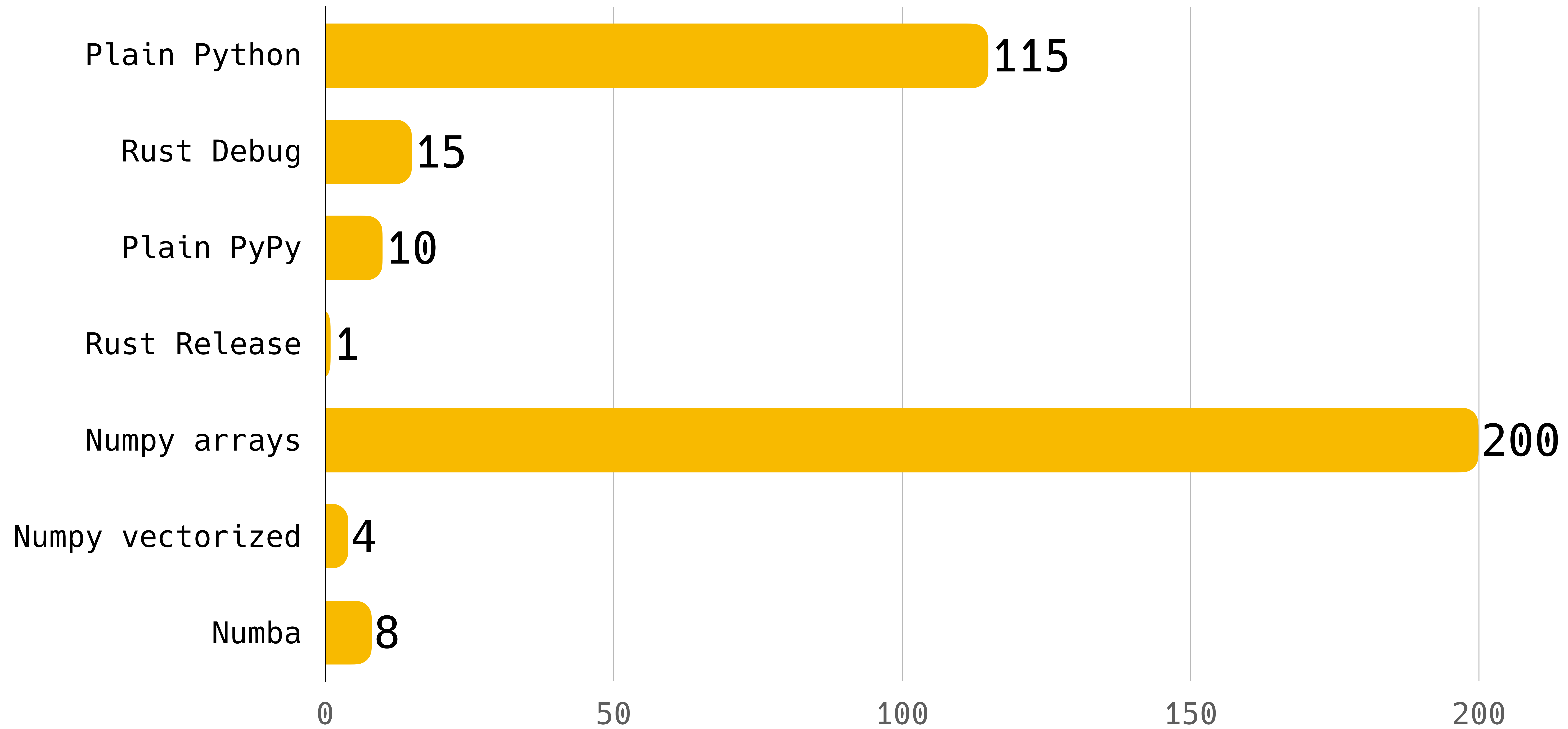
Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
Numpy arrays
200
Numpy vectorized
4
0
50
100
150
200

Numba

just-in-time компиляция

- добавили декоратор @njit
- einsum не поддерживается, пришлось написать dot_batch:

```
1.@njit
2.def dot_batch(a: npt.NDArray, b: npt.NDArray) -> npt.NDArray:
3.    dotted = np.zeros(len(a))
4.    for i in range(len(a)):
5.        dotted[i] = np.dot(a[i], b[i])
6.    return dotted
7.
8.
```


Plain Python
115
Rust Debug
15
Plain PyPy
10
Rust Release
1
Numpy arrays
200
Numpy vectorized
4
Numba
8
0
50
100
150
200

Что дальше?

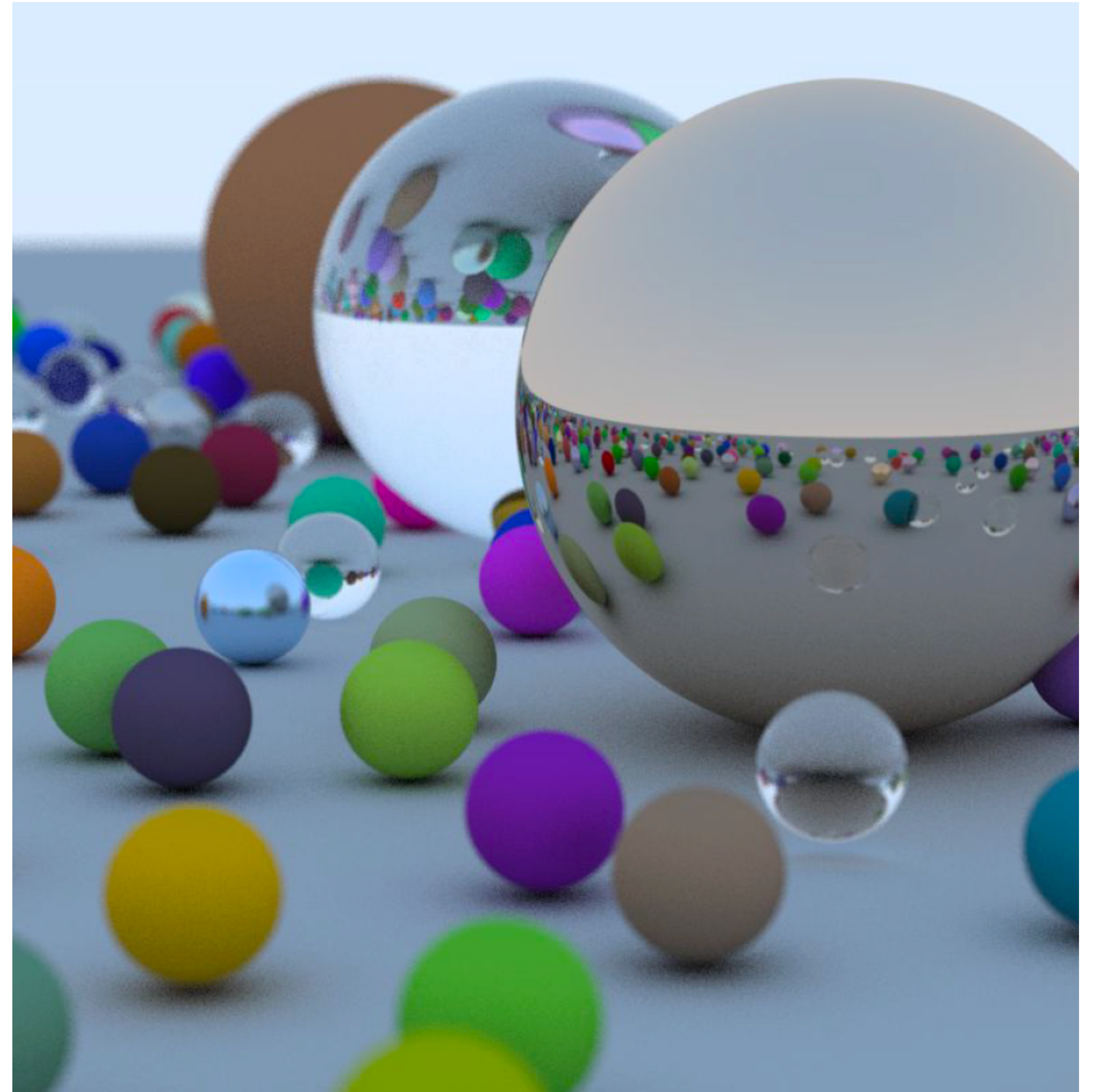
Куда еще можно покопать

и это не выглядит очень сложным

- раскидать по ядрам
- запустить на GPU
- добавить кубики, конусы, ...
- добавить преломления, затухание в среде, ...
- карты окружения, текстуры, карты нормалей
- квантовые эффекты (?)
- репозиторий тут: github.com/direvius/rt ;)

Список литературы

- серия статей Raytracing in a weekend clck.ru/36ZW8C
- видео об отражениях от Nayar clck.ru/36ZW9k
- книга Fluent Python (Luciano Ramalho)



Спасибо за внимание!