

# Фотореалистичный рендеринг *(aka raytracing)*

Лекция 1: Введение в курс. Метод трассировки лучей.  
Пересечение луча с геометрическими примитивами.

---

2024

- Лисица Никита Игоревич (Яндекс)
- lisyarus@gmail.com
- +7 (952) 276-70-50

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
- Практики

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
- Практики
  - Слайды с заданиями в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
- Практики
  - Слайды с заданиями в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
  - Каждая практика – одно большое задание

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
- Практики
  - Слайды с заданиями в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
  - Каждая практика – одно большое задание
  - Сдача автотестированием + код-ревью

# Как устроен курс

- Лекции
  - Слайды в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
- Практики
  - Слайды с заданиями в репозитории [github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf](https://github.com/lisyarus/raytracing-course-slides/tree/trunk/pdf)
  - Каждая практика – одно большое задание
  - Сдача автотестированием + код-ревью
  - Каждая практика основывается на коде предыдущей практики!

# Баллы

## Баллы

- Все баллы получаются за практические задания

## Баллы

- Все баллы получаются за практические задания
  - до 9 баллов – 1ая практика
  - до 10 баллов – 2ая практика
  - ...
  - до 16 баллов – 8ая практика

## Баллы

- Все баллы получаются за практические задания
  - до 9 баллов – 1ая практика
  - до 10 баллов – 2ая практика
  - ...
  - до 16 баллов – 8ая практика
- В сумме – **100 баллов**

## Баллы

- Дедлайн для практик – 1 неделя, т.е. 23:59 следующего понедельника

# Баллы

- Дедлайн для практик – 1 неделя, т.е. 23:59 следующего понедельника
  - На 1ую практику дедлайн – 3 недели
  - На 2ую практику дедлайн – 2 недели

## Баллы

- Дедлайн для практик – 1 неделя, т.е. 23:59 следующего понедельника
  - На 1ую практику дедлайн – 3 недели
  - На 2ую практику дедлайн – 2 недели
- При сдаче после дедлайна баллы делятся **пополам** (с округлением в большую сторону)

# Оценка за курс

## Оценка за курс

- Зачет: **50 и более** баллов

## Оценка за курс

- Зачет: **50 и более** баллов
- Экзамен:
  - 50-59 баллов: E
  - 60-69 баллов: D
  - 70-79 баллов: C
  - 80-89 баллов: B
  - 90-100 баллов: A

# Автотестирование

- Каждая практика – программа, по описанию входной сцены генерирующая картинку ('рендер')

# Автотестирование

- Каждая практика – программа, по описанию входной сцены генерирующая картинку ('рендер')
- Автотестированием занимается телеграм-бот `mkn-raytracing-2024-bot`

# Автотестирование

- Каждая практика – программа, по описанию входной сцены генерирующая картинку ('рендер')
- Автотестированием занимается телеграм-бот `mkn-raytracing-2024-bot`
- Он запускает вашу программу на 5 случайных сценах и сравнивает картинку с референсом

# Автотестирование

- Каждая практика – программа, по описанию входной сцены генерирующая картинку ('рендер')
- Автотестированием занимается телеграм-бот `mkn-raytracing-2024-bot`
- Он запускает вашу программу на 5 случайных сценах и сравнивает картинку с референсом
- Баллы вычисляются на основе худшего процента совпадения вашей картинки и картинки-референса

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)
- **N.B.:** Наши программы будут довольно медленными, так что лучше выбрать язык пошустрее

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)
- **N.B.:** Наши программы будут довольно медленными, так что лучше выбрать язык пошустрее
- Нам нужно будет
  - Читать и писать в текстовые и бинарные файлы

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)
- **N.B.:** Наши программы будут довольно медленными, так что лучше выбрать язык пошустрее
- Нам нужно будет
  - Читать и писать в текстовые и бинарные файлы
  - Работать с векторами и матрицами, – можно взять готовую библиотеку (напр. `glm`), а можно написать все типы и операции самим (нам нужно совсем немного)

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)
- **N.B.:** Наши программы будут довольно медленными, так что лучше выбрать язык пошустрее
- Нам нужно будет
  - Читать и писать в текстовые и бинарные файлы
  - Работать с векторами и матрицами, – можно взять готовую библиотеку (напр. `glm`), а можно написать все типы и операции самим (нам нужно совсем немного)
  - Генерировать случайные числа

# Автотестирование

- Можно использовать любой язык программирования (если он не поддерживается – напишите мне, я постараюсь его добавить)
- **N.B.:** Наши программы будут довольно медленными, так что лучше выбрать язык пошустрее
- Нам нужно будет
  - Читать и писать в текстовые и бинарные файлы
  - Работать с векторами и матрицами, – можно взять готовую библиотеку (напр. `glm`), а можно написать все типы и операции самим (нам нужно совсем немного)
  - Генерировать случайные числа
  - Оперировать древовидными структурами данных

# Автотестирование

- Боту нужно послать команду вида

```
/submit practice1 https://github.com/vanya/practice1.git  
[ path-inside-repo [ branch ] ]
```

# Автотестирование

- Боту нужно послать команду вида

```
/submit practice1 https://github.com/vanya/practice1.git  
[ path-inside-repo [ branch ] ]
```

- Можно делать все задания в одном репозитории или в разных, как хотите

# Автотестирование

- Боту нужно послать команду вида

```
/submit practice1 https://github.com/vanya/practice1.git  
[ path-inside-repo [ branch ] ]
```

- Можно делать все задания в одном репозитории или в разных, как хотите
- В репозитории по указанному пути должны быть скрипты **build.sh** и **run.sh**:
  - **build.sh** запускается один раз для сборки проекта
  - **run.sh** запускается для каждого теста, он должен принимать пути до входного и выходного файла

# Автотестирование

Пример для проекта на C++ и CMake:

- build.sh

```
#!/usr/bin/env bash

mkdir build
cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build .
```

- run.sh

```
#!/usr/bin/env bash

./build/solution "$1" "$2"
```

# Пререквизиты

# Пререквизиты

- Программирование

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.

# Пререквизиты

- Программирование
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде
- **Математика**

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде
- **Математика**
  - Линейная алгебра (векторы, матрицы, умножение матриц, линейные системы, ортогональность)

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде
- **Математика**
  - Линейная алгебра (векторы, матрицы, умножение матриц, линейные системы, ортогональность)
  - Аналитическая геометрия (координаты, уравнения кривых и поверхностей)

# Пререквизиты

- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде
- **Математика**
  - Линейная алгебра (векторы, матрицы, умножение матриц, линейные системы, ортогональность)
  - Аналитическая геометрия (координаты, уравнения кривых и поверхностей)
  - Анализ (производные, интегралы, трансцендентные функции)

# Пререквизиты

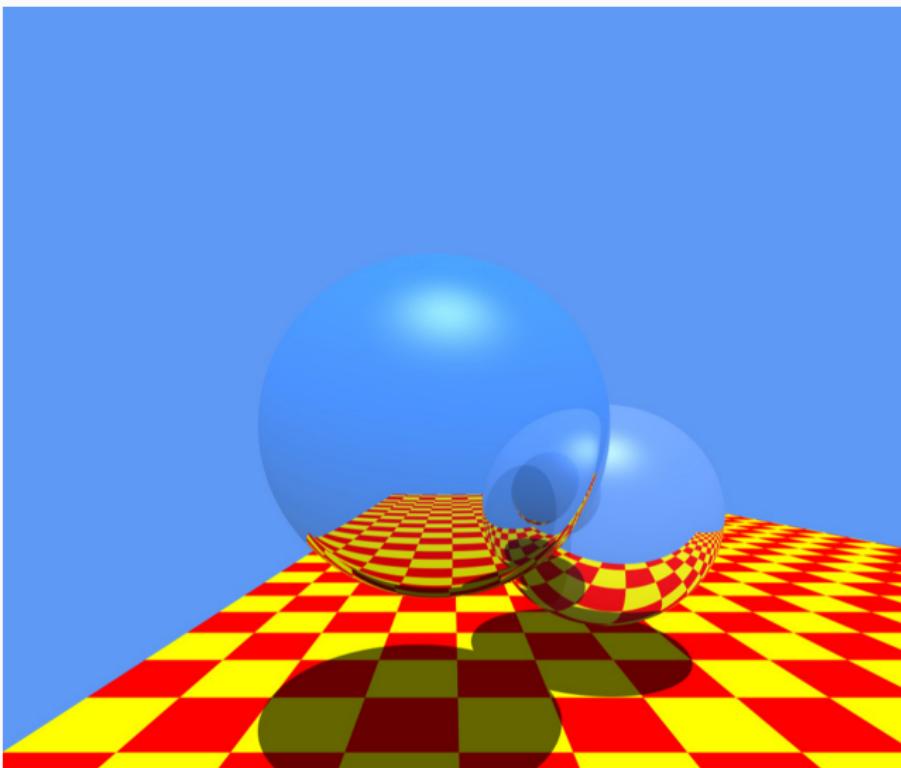
- **Программирование**
  - Основы программирования – циклы, массивы, функции, структуры, и т.п.
  - Какой-нибудь язык программирования
  - Компилировать и запускать программы в удобной вам среде
- **Математика**
  - Линейная алгебра (векторы, матрицы, умножение матриц, линейные системы, ортогональность)
  - Аналитическая геометрия (координаты, уравнения кривых и поверхностей)
  - Анализ (производные, интегралы, трансцендентные функции)
  - Основы статистики

# Примерный план курса

# Примерный план курса

- Лекции 1-3: Whitted-style raytracing
  - Трассировка лучей без решения уравнения рендеринга

# Whitted-style raytracing (1979)



# Примерный план курса

- Лекции 1-3: Whitted-style raytracing
  - Трассировка лучей без решения уравнения рендеринга

# Примерный план курса

- Лекции 1-3: Whitted-style raytracing
  - Трассировка лучей без решения уравнения рендеринга
- Лекции 4-6: Monte-Carlo pathtracing
  - Теория распространения света
  - Модели материалов
  - Решение уравнение рендеринга
  - Оптимизация

# Monte-Carlo pathtracing



# Примерный план курса

- Лекции 1-3: Whitted-style raytracing
  - Трассировка лучей без решения уравнения рендеринга
- Лекции 4-6: Monte-Carlo pathtracing
  - Теория распространения света
  - Модели материалов
  - Решение уравнение рендеринга
  - Оптимизация

# Примерный план курса

- Лекции 1-3: Whitted-style raytracing
  - Трассировка лучей без решения уравнения рендеринга
- Лекции 4-6: Monte-Carlo pathtracing
  - Теория распространения света
  - Модели материалов
  - Решение уравнение рендеринга
  - Оптимизация
- Лекции 7-8: Красивые картинки
  - Формат сцен glTF
  - Physically-based материалы
  - Тестуры
  - Material mapping

# Красивые картинки



# Что такое компьютерная графика?

# Что такое компьютерная графика?

- Кинематограф, мультипликация

# The Matrix Revolutions (2003)



# Avatar (2009)



# The Avengers (2012)



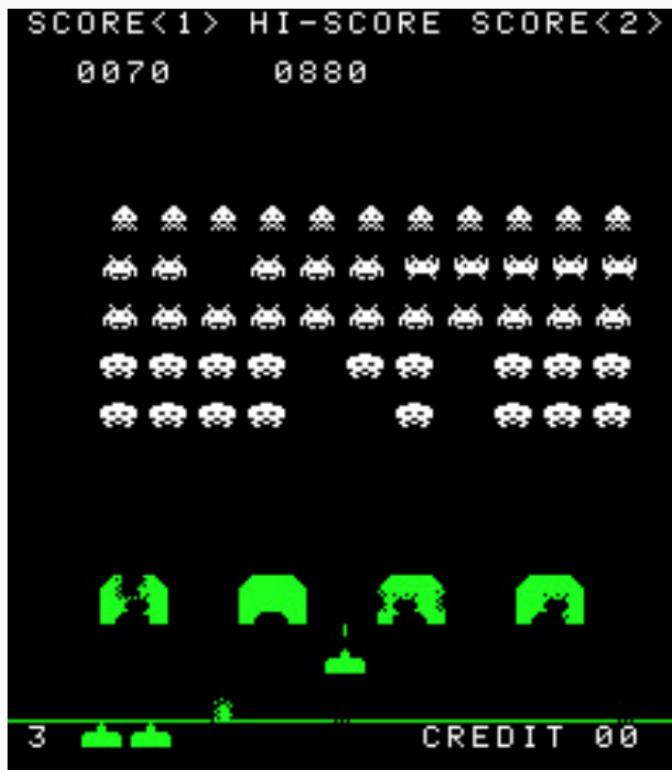
# Klaus (2019)



# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры

# Space Invaders (1978)



# Doom (1993)



# Grand Theft Auto: Vice City (2002)



# Civilization V (2010)



# The Witcher 3: Wild Hunt (2015)



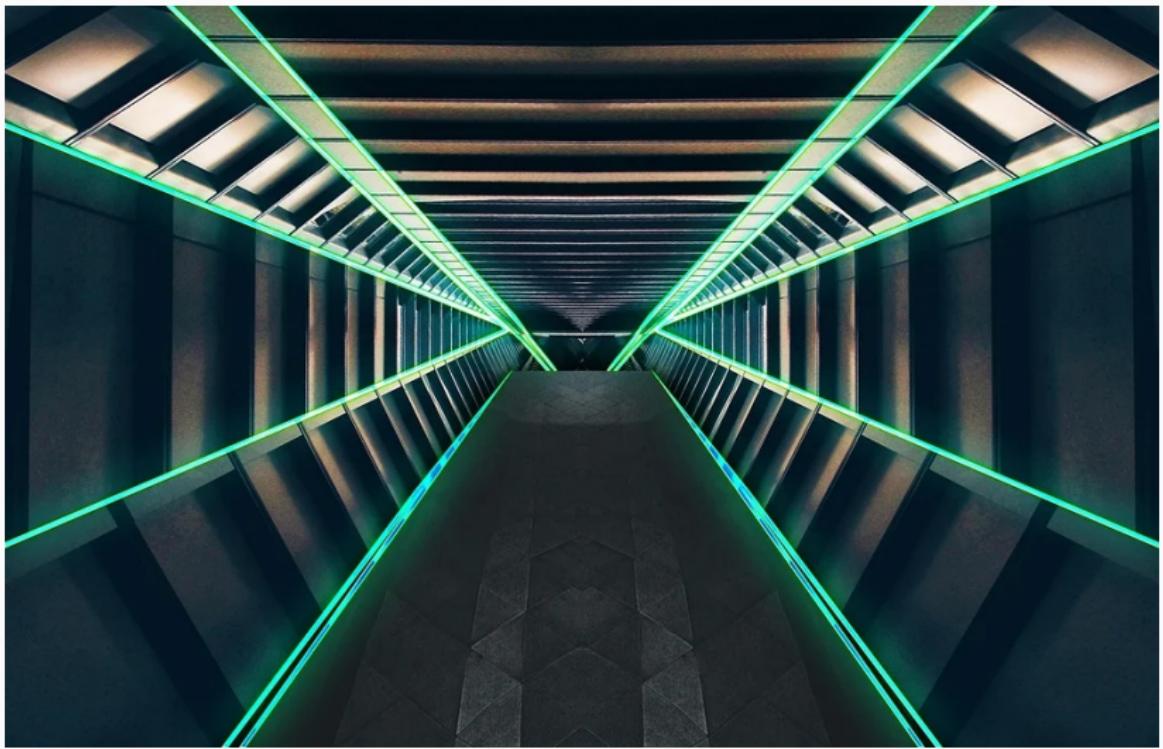
# Cyberpunk 2077 (2020)

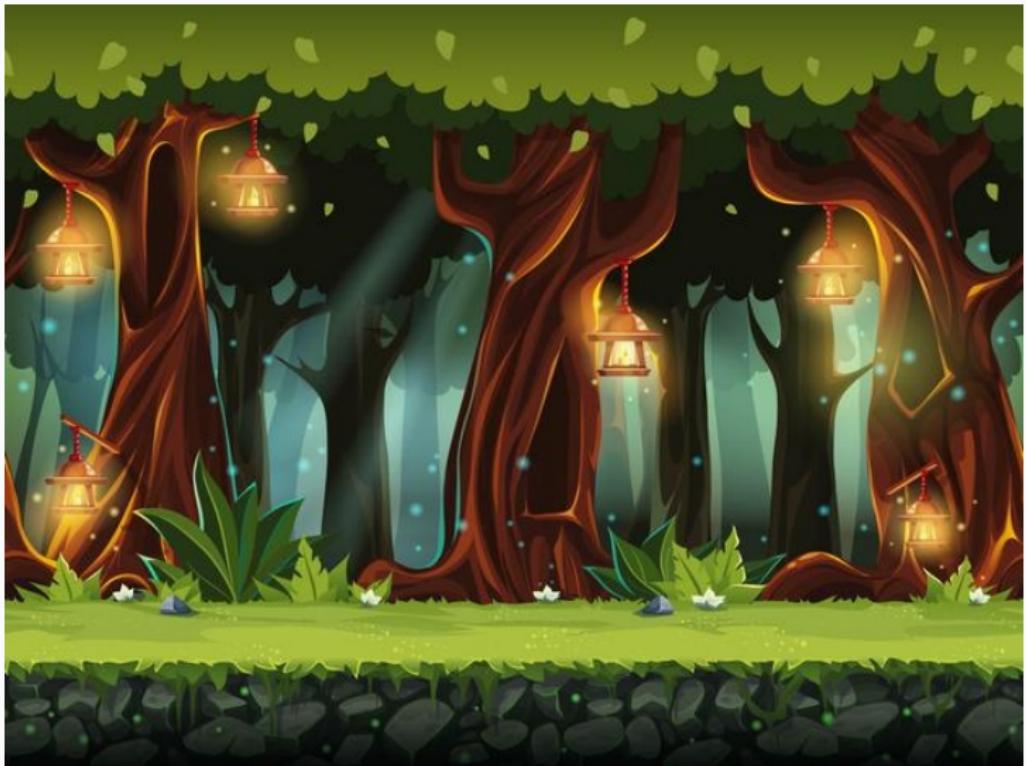


# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art







# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art
- Графический интерфейс

# Mac OS Catalina



# Windows 10

A System accent color: #0078D4

Buttons	Calendar Date Picker	Combo box	Textbox
Enabled button	Label title mm/dd/yyyy	Label title Placeholder text	Label title Placeholder text
Disabled button	Hover mm/dd/yyyy	Hover Placeholder text	Hover Placeholder text
Toggle button	Disabled mm/dd/yyyy	Disabled Placeholder text	Disabled Placeholder text
Checkbox	Radio button	Combo box	Textbox
<input type="checkbox"/> Unchecked	<input type="radio"/> Unchecked	Microsoft	Typing This is text.]
<input checked="" type="checkbox"/> Checked	<input checked="" type="radio"/> Checked	Windows	Password *****
<input type="checkbox"/> Third state	<input checked="" type="radio"/> Checked	Office	
<input checked="" type="checkbox"/> Disabled			
Toggle switch			
<input type="checkbox"/> Off		<input type="checkbox"/> Disabled Off	
<input checked="" type="checkbox"/> On		<input checked="" type="checkbox"/> Disabled On	

**Buttons**

Enabled button  
Disabled button  
Toggle button

**Checkbox**

Unchecked  
 Checked  
 Third state  
 Disabled

**Radio button**

Unchecked  
 Checked  
 Disabled

**Calendar Date Picker**

Label title  
mm/dd/yyyy

Hover  
mm/dd/yyyy

Disabled  
mm/dd/yyyy

**Combo box**

Label title  
Placeholder text

Hover  
Placeholder text

Disabled  
Placeholder text

**Textbox**

Label title  
Placeholder text

Hover  
Placeholder text

Disabled  
Placeholder text

Typing  
This is text.]

Password  
\*\*\*\*\*

**Toggle switch**

Off  Disabled Off

On  Disabled On

# Europa Universalis 4

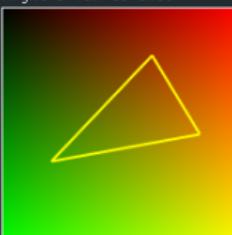


# Dear ImGui

▼ Example: Custom rendering

Clear Undo

Left-click and drag to add  
Right-click to undo



E028E4: 00 00 F8 44 00 40 7E 44 ...D.@@D  
E028EC: 06 33 88 3C 00 00 A8 40 ...<...@  
E028F4: 18 BA DE 00 00 BA DE 00 .....  
E028FC: 9A 99 99 3E 00 00 C8 40 ...>...@  
E02984: 00 00 C0 40 02 01 00 00 ...@...  
E0298C: 07 01 00 00 06 01 00 00 .....  
E02914: 09 01 00 00 08 01 00 00 .....  
E0291C: 0A 01 00 00 0B 01 00 00 .....  
E02924: 0C 01 00 00 0D 01 00 00 .....  
E0292C: 05 01 00 00 03 01 00 00 .....  
E02934: 01 01 00 00 00 01 00 00 .....  
E0293C: 41 00 00 00 43 00 00 00 A...C...  
E02944: 56 00 00 00 58 00 00 00 V...X...Console  
E0294C: 59 00 00 00 5A 00 00 00 Y...Z...  
E02954: 00 00 00 3E CD CC 4C 3D ...>URL implements a console with basic coloring,  
E0295C: 00 00 00 00 A4 28 E0 00 ...let!(); and history. A more elaborate implementation may  
support storing entries along with extra data such as timestamp,

8 rows Range E028E4..E03C93 Iter: etc.

▼ Example: Memory Editor

Address	Value
E028E4	00 00 F8 44 00 40 7E 44
E028EC	06 33 88 3C 00 00 A8 40
E028F4	18 BA DE 00 00 BA DE 00
E028FC	9A 99 99 3E 00 00 C8 40
E02984	00 00 C0 40 02 01 00 00
E0298C	07 01 00 00 06 01 00 00
E02914	09 01 00 00 08 01 00 00
E0291C	0A 01 00 00 0B 01 00 00
E02924	0C 01 00 00 0D 01 00 00
E0292C	05 01 00 00 03 01 00 00
E02934	01 01 00 00 00 01 00 00
E0293C	41 00 00 00 43 00 00 00
E02944	56 00 00 00 58 00 00 00
E0294C	59 00 00 00 5A 00 00 00
E02954	00 00 00 3E CD CC 4C 3D
E0295C	00 00 00 00 A4 28 E0 00

use functions such as IsItemHovered() on

AAA BBB CCC EEE DDD LEVERAGE BUZZWORD

ACTION REACTION

Text Baseline Alignment  
Scrolling

Enter 'HELP' for help, press TAB to use text completion.

Add Dummy Text Add Dummy Error Clear

Filter ("incl,-excl") ("error")

0 some text  
some more text  
display very important message here!  
[error] something went wrong  
Possible matches:  
- HELP  
- HISTORY  
# HELP  
Commands:  
- HELP  
- HISTORY  
- CLEAR  
- CLASSIFY  
# hello, imgui world!  
Unknown command: 'hello, imgui world!'

hello, imgui world! Input

▼ Example: Layout

File

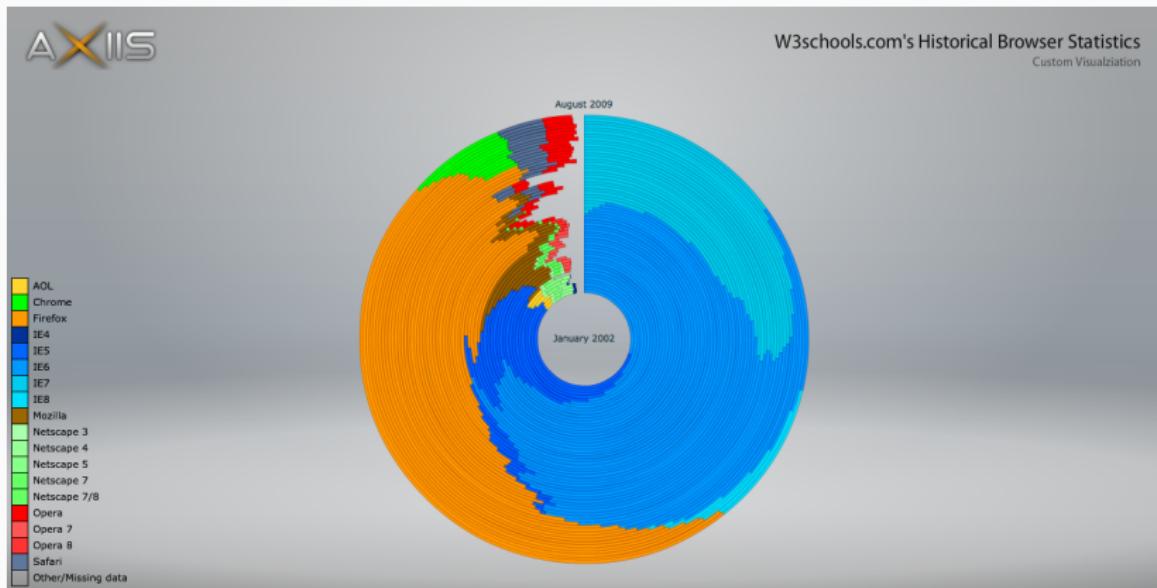
MyObject 0  
MyObject 1  
MyObject 2  
MyObject 3  
MyObject 4  
MyObject 5  
MyObject 6  
MyObject 7  
MyObject 8  
MyObject 9  
MyObject 10  
MyObject 11  
MyObject 12  
MyObject 13

Revert Save

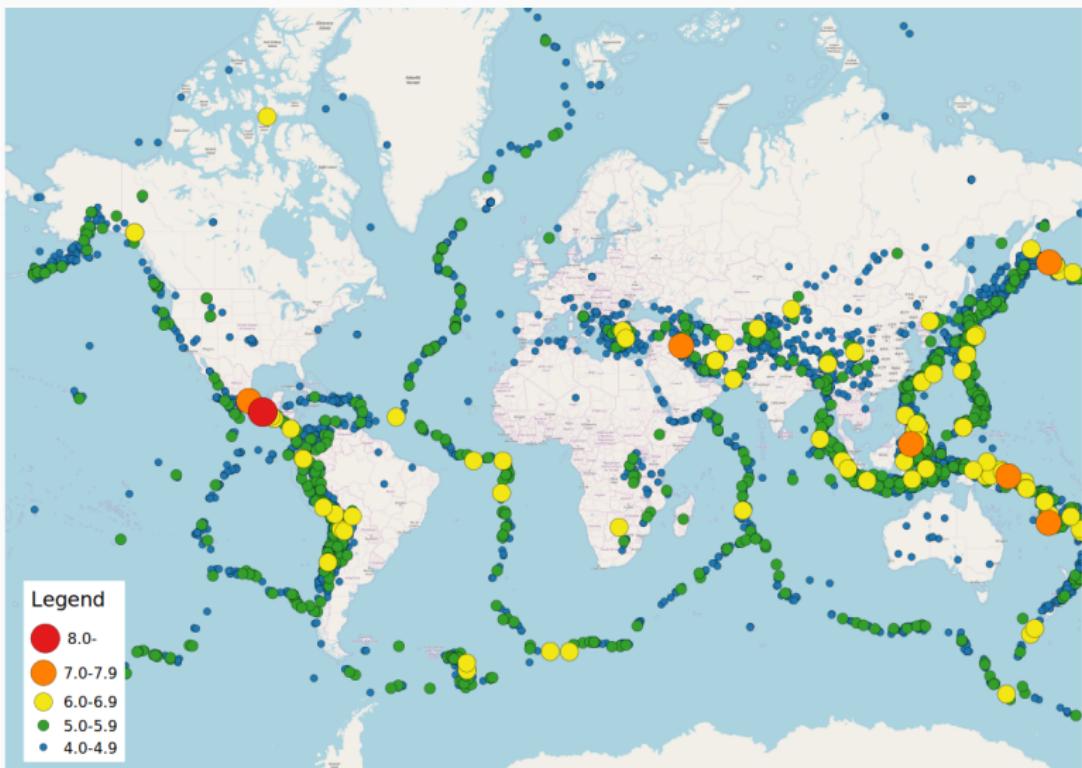
# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art
- Графический интерфейс
- Визуализация данных

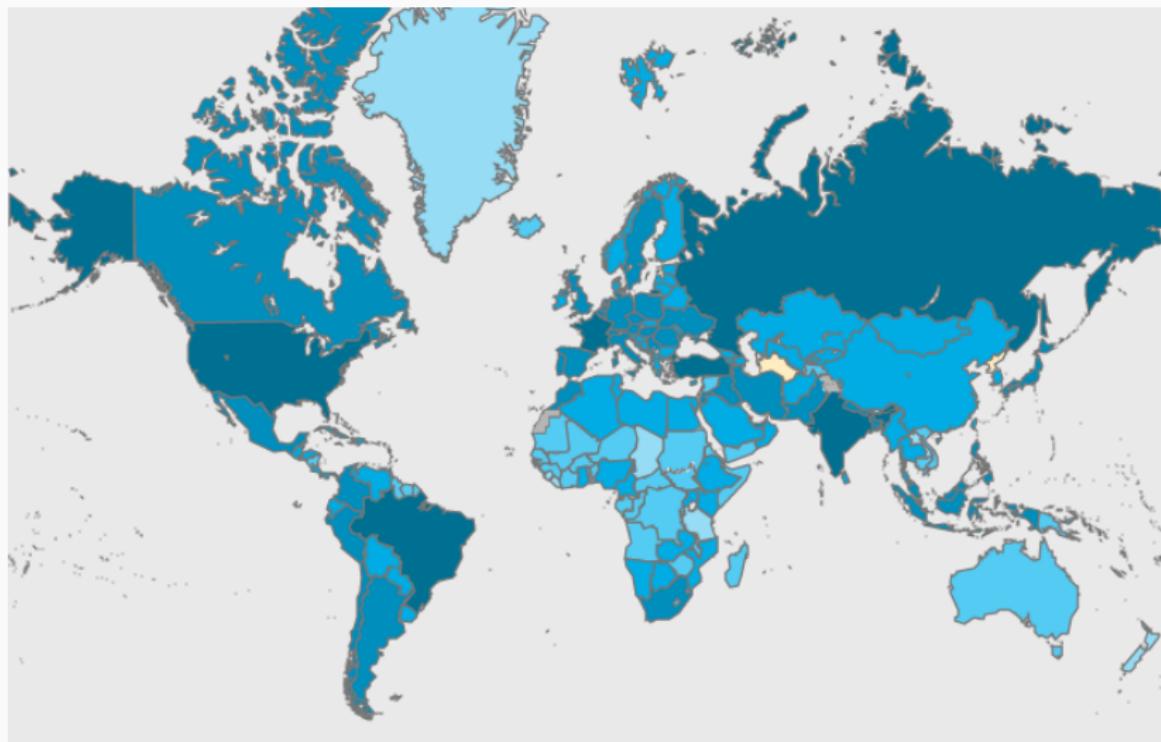
# Популярность браузеров в 2002-2009



# Карта землетрясений



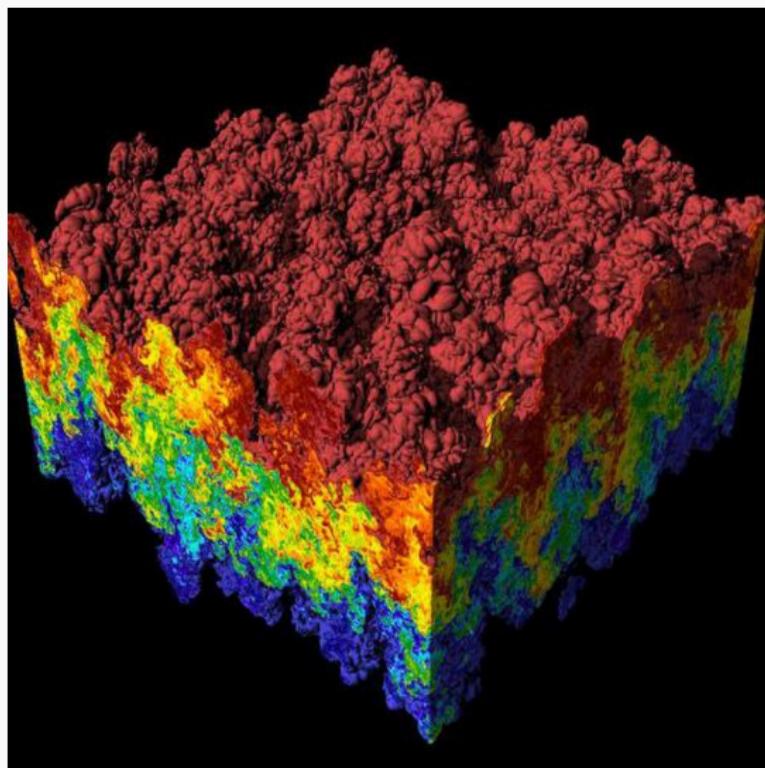
# Количество случаев заражения COVID-19



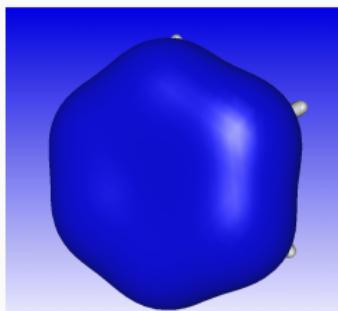
# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art
- Графический интерфейс
- Визуализация данных
- Научная визуализация

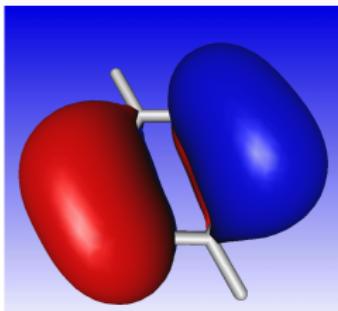
# Неустойчивость Рэлея – Тейлора



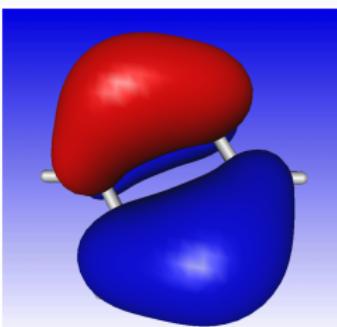
# Молекулярные орбитали бензола



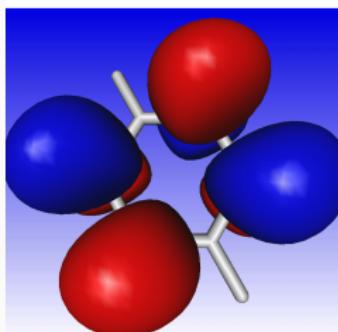
16



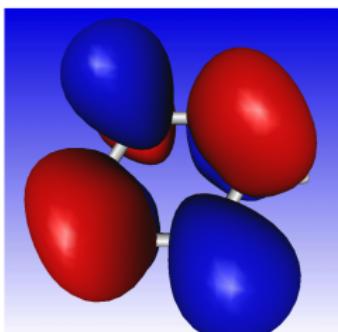
20



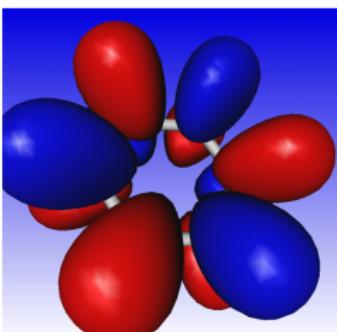
21



22

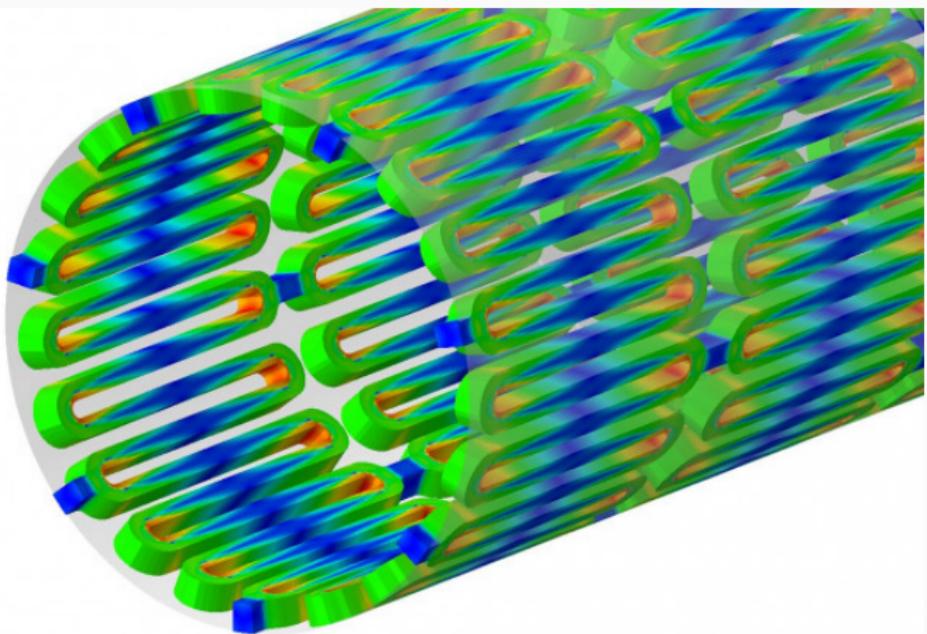


23



30

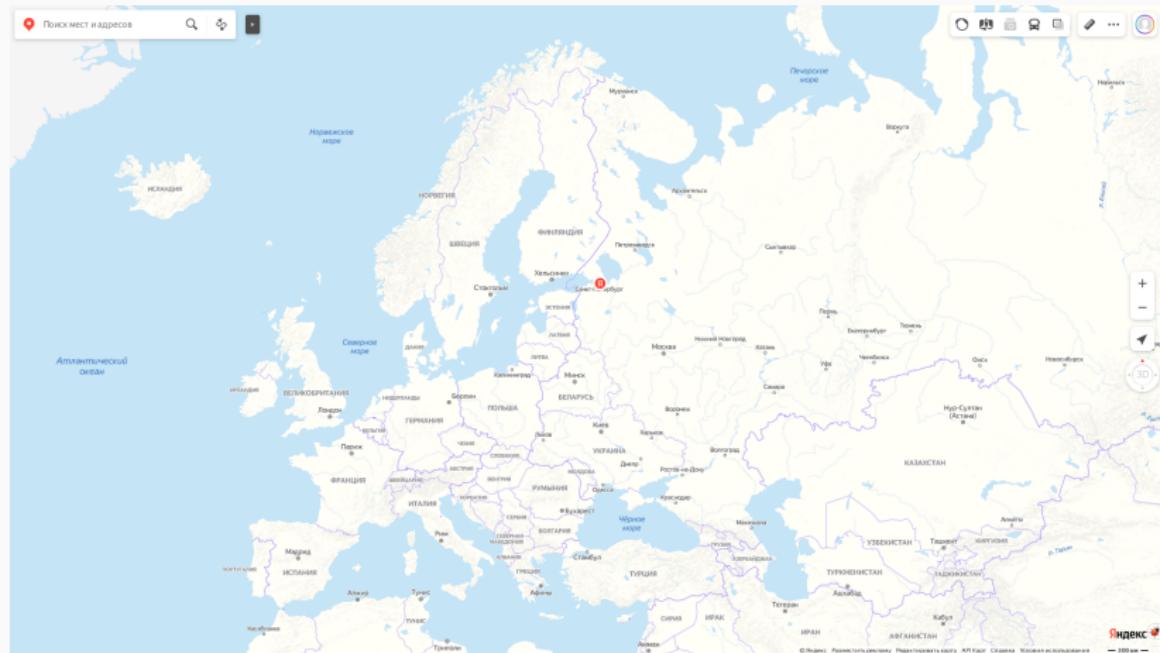
# Симуляция напряжений в стене методом конечных элементов



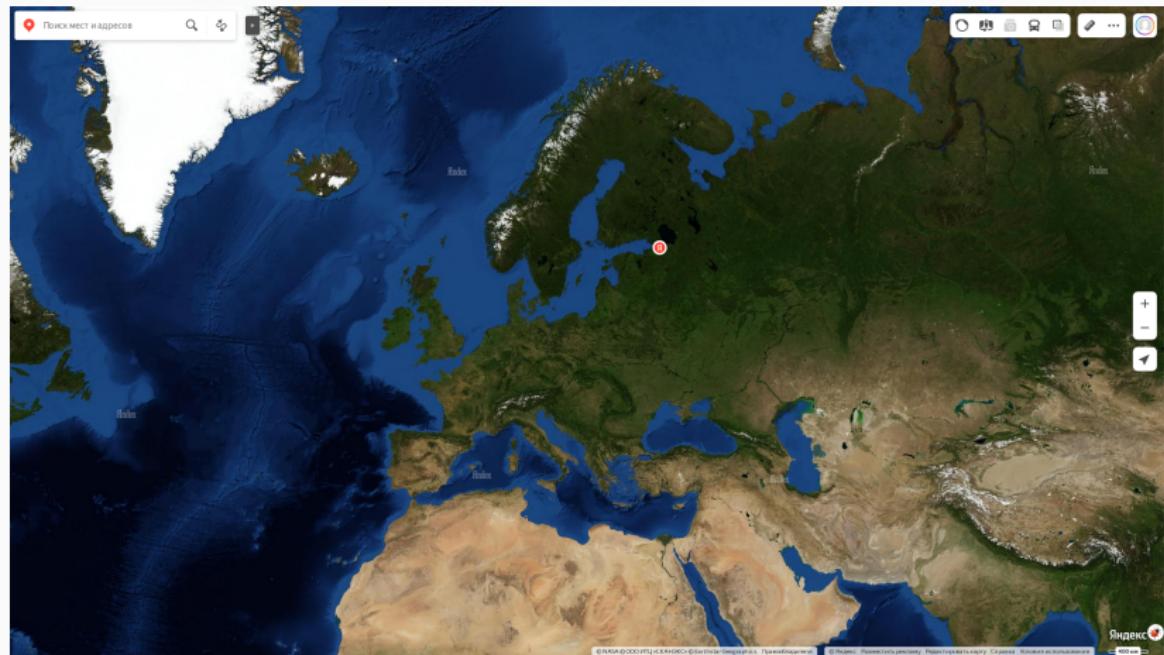
# Что такое компьютерная графика?

- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art
- Графический интерфейс
- Визуализация данных
- Научная визуализация
- Карты

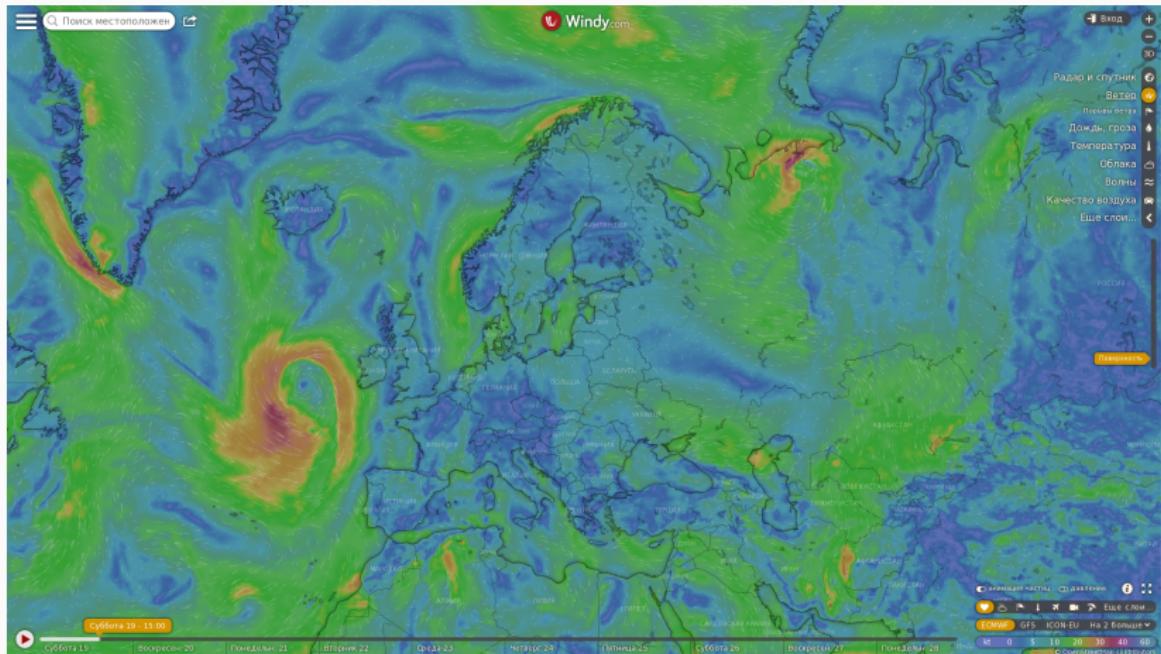
## Схематическая карта



# Спутниковая карта



## Карта погоды



# Что такое компьютерная графика?

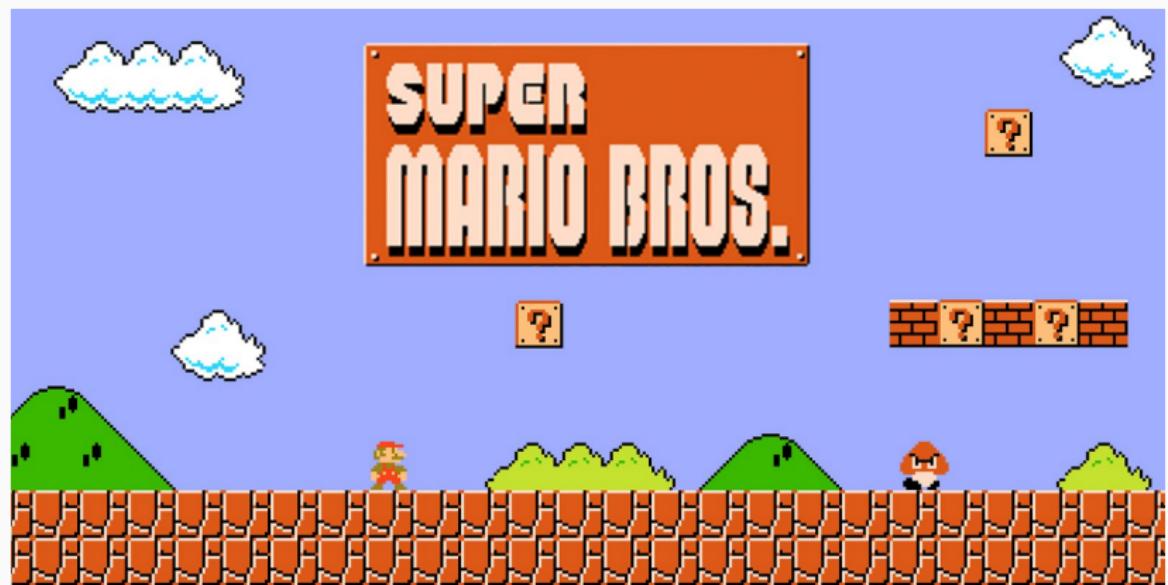
- Кинематограф, мультипликация
- Компьютерные игры
- Рисунки, concept art
- Графический интерфейс
- Визуализация данных
- Научная визуализация
- Карты
- И т.д.

# Грубая и неточная классификация

## Грубая и неточная классификация

- 2D / 3D

# Super Mario Bros. (1983) - 2D



# Red Dead Redemption 2 (2018) - 3D



## Грубая и неточная классификация

- 2D / 3D

## Грубая и неточная классификация

- 2D / 2.5D / 3D

# Civilization III (2001) - 2.5D



## Грубая и неточная классификация

- 2D / 2.5D / 3D

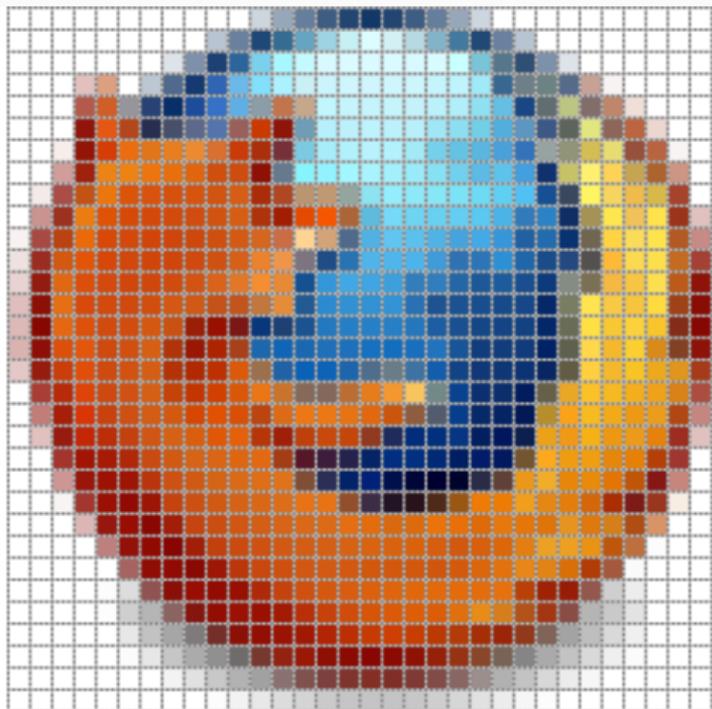
## Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая

# Векторная графика



# Растровая графика



## Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая

## Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / offline

## Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline

## Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная

# Грубая и неточная классификация

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / **3D**
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

# Чем мы будем заниматься?

- 2D / 2.5D / 3D
- Векторная / растровая
- Realtime / near real-time / offline
- Фотореалистичная / стилизованная
- CPU / GPU

Где это пригодится?

## Где это пригодится?

- Разработка профессиональных движков  
фотореалистичного рендеринга

Чем мы не будем заниматься?

## Чем мы не будем заниматься?

- Учиться рисовать / моделировать и анимировать объекты / etc.

# Чем мы не будем заниматься?

- Учиться рисовать / моделировать и анимировать объекты / etc.
  - Красивая картинка – движок + данные (текстуры, модели, частицы, etc, – assets)
  - Курс про [движок](#)

# Два способа рендеринга

- Рендеринг – генерация изображения по входной сцене

# Два способа рендеринга

- Рендеринг – генерация изображения по входной сцене
- Первый способ: *растеризация*

# Два способа рендеринга

- Рендеринг – генерация изображения по входной сцене
- Первый способ: *растеризация*
  - Разобьём сцену на максимально простые куски (треугольники)
  - Вычислим, в какие пиксели на экране попал каждый треугольник
  - Сделаем тонну трюков, чтобы обработать наложение объектов, освещение, и т.д.
  - + Очень быстро работает
  - + Требует только простейших операций
  - + Хорошо ложится на GPU
  - - Очень сложно реализовать нетривиальные эффекты

# Два способа рендеринга

- Второй способ: *трассировка лучей/путей*

# Два способа рендеринга

- Второй способ: *трассировка лучей/путей*
  - Разобьём сцену на объекты
  - Для каждого пикселя построим луч из камеры в этот пиксель
  - Вычислим, что пересёк этот луч
    - – Работает медленнее из-за поиска пересечений
    - – Плохо ложится на GPU из-за необходимости передачи сложных структур данных
  - + С помощью дополнительных лучей легко реализовать почти любые эффекты, связанные с распространением света

# Два способа рендеринга

- Второй способ: *трассировка лучей/путей*
  - Разобьём сцену на объекты
  - Для каждого пикселя построим луч из камеры в этот пиксель
  - Вычислим, что пересёк этот луч
    - – Работает медленнее из-за поиска пересечений
    - – Плохо ложится на GPU из-за необходимости передачи сложных структур данных
    - + С помощью дополнительных лучей легко реализовать почти любые эффекты, связанные с распространением света
- Это то, чем мы будем заниматься

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит
- 3. Задать размеры изображения

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит
- 3. Задать размеры изображения
- 4. Для каждого пикселя изображения, построить луч из камеры в этот пиксель

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит
- 3. Задать размеры изображения
- 4. Для каждого пикселя изображения, построить луч из камеры в этот пиксель
- 5. Найти ближайшее пересечение луча с объектами сцены

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит
- 3. Задать размеры изображения
- 4. Для каждого пикселя изображения, построить луч из камеры в этот пиксель
- 5. Найти ближайшее пересечение луча с объектами сцены
- 6. В качестве цвета пикселя взять цвет объекта пересечения

# Трассировка лучей

Общий план рисования сцены трассировкой лучей:

- 1. Описать сцену: форму объектов и их материал (напр. цвет)
- 2. Описать камеру: где она находится, куда смотрит
- 3. Задать размеры изображения
- 4. Для каждого пикселя изображения, построить луч из камеры в этот пиксель
- 5. Найти ближайшее пересечение луча с объектами сцены
- 6. В качестве цвета пикселя взять цвет объекта пересечения
- 7. Сохранить полученную картинку

## Описание сцены

- Поначалу мы будем использовать максимально простой формат сцен, описанный в слайдах практики

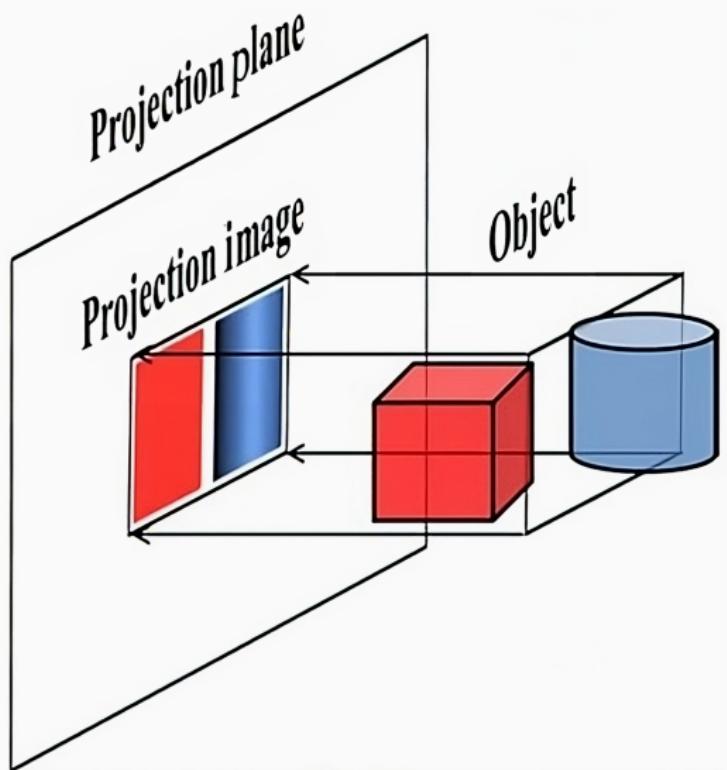
## Описание сцены

- Поначалу мы будем использовать максимально простой формат сцен, описанный в слайдах практики
- Ближе к концу курса перейдём к более серьёзному формату, скорее всего – glTF

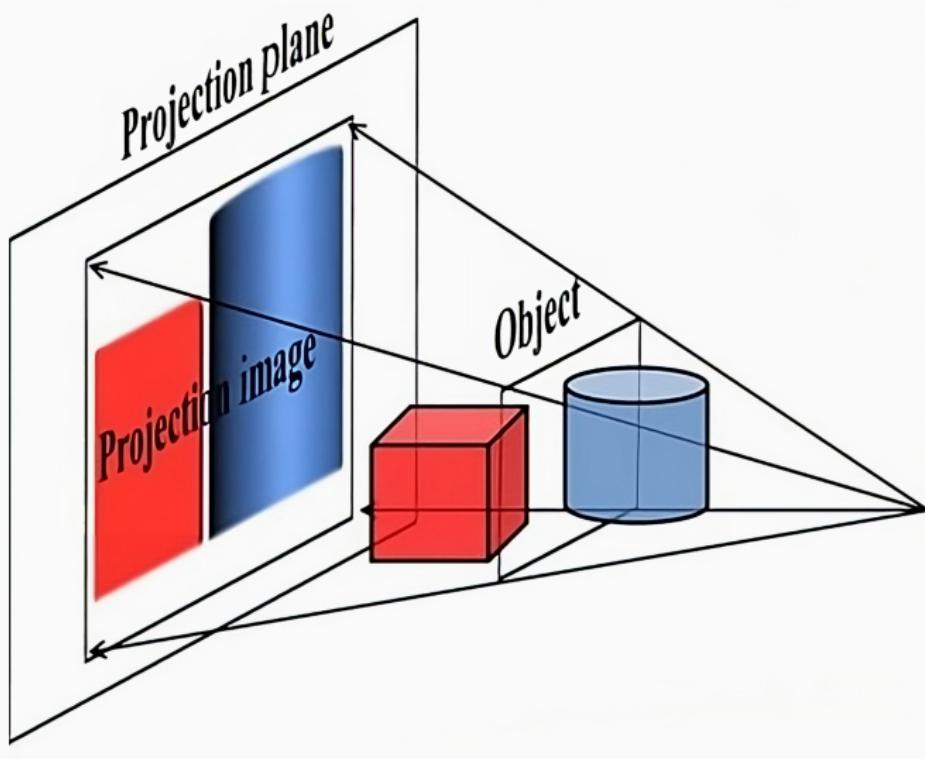
## Описание камеры

- Обычно в компьютерной графике используют два типа камеры (*проекции*): ортографическую и перспективную

## Ортографическая камера



## Перспективная камера



## Описание камеры

- Обычно в компьютерной графике используют два типа камеры (*проекции*): ортографическую и перспективную

## Описание камеры

- Обычно в компьютерной графике используют два типа камеры (*проекции*): ортографическую и перспективную
- Мы будем использовать перспективную камеру

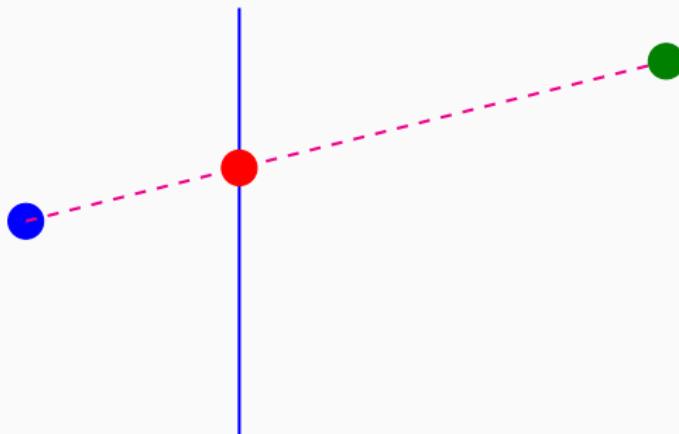
# Перспективная проекция

- Есть **центр проекции** и **плоскость проекции**
- 



# Перспективная проекция

- Есть **центр проекции** и **плоскость проекции**
- **Проекция точки** – пересечение **прямой**, проходящей через эту **точку** и **центр проекции**, с **плоскостью проекции**



## Описание камеры

- Позиция камеры: точка, из которой камера смотрит

## Описание камеры

- Позиция камеры: точка, из которой камера смотрит
- Оси камеры (вперёд, вправо, вверх): векторы, задающие ориентацию камеры

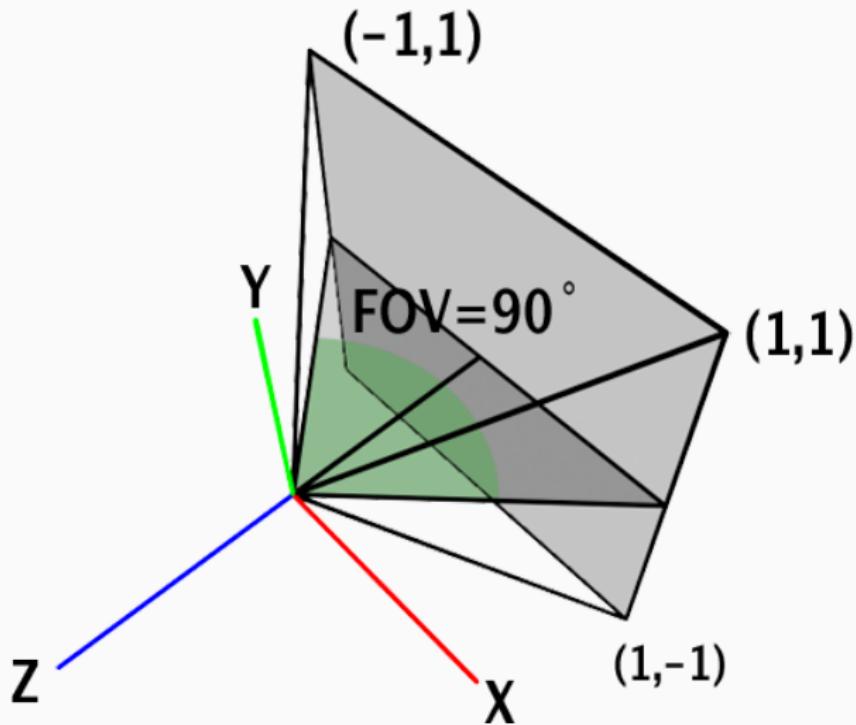
## Описание камеры

- Позиция камеры: точка, из которой камера смотрит
- Оси камеры (вперёд, вправо, вверх): векторы, задающие ориентацию камеры
  - **N.B.** Будем считать, что оси в таком порядке задают левую систему координат

## Описание камеры

- Позиция камеры: точка, из которой камера смотрит
- Оси камеры (вперёд, вправо, вверх): векторы, задающие ориентацию камеры
  - **N.B.** Будем считать, что оси в таком порядке задают левую систему координат
- Угол обзора камеры по ширине и высоте (*FOV – field of view*)

## Углы обзора камеры



## Углы обзора камеры

- Угол обзора по ширине  $fov_x$  – угол между левой и правой границами видимой области

## Углы обзора камеры

- Угол обзора по ширине  $fov_x$  – угол между левой и правой границами видимой области
- Угол обзора по высоте  $fov_y$  – угол между верхней и нижней границами видимой области

## Вычисление луча из камеры

- Хотим вычислить луч из камеры в пиксель с (целыми) координатами  $(P_X, P_Y)$

## Вычисление луча из камеры

- Хотим вычислить луч из камеры в пиксель с (целыми) координатами  $(P_X, P_Y)$
- Луч задаётся началом и направлением

## Вычисление луча из камеры

- Хотим вычислить луч из камеры в пиксель с (целыми) координатами  $(P_X, P_Y)$
- Луч задаётся началом и направлением
- Начало луча  $O$  это позиция камеры

## Вычисление луча из камеры

- Хотим вычислить луч из камеры в пиксель с (целыми) координатами  $(P_X, P_Y)$
- Луч задаётся началом и направлением
- Начало луча  $O$  это позиция камеры
- Направление луча рассчитывается из координат пикселя, углов обзора камеры и её ориентации

## Вычисление луча из камеры

- Игнорируя ориентацию камеры, представим, что она находится в начале координат и её оси совпадают с осями координат (камера смотрит в направлении оси Z)

## Вычисление луча из камеры

- Игнорируя ориентацию камеры, представим, что она находится в начале координат и её оси совпадают с осями координат (камера смотрит в направлении оси Z)
- Возьмём плоскость  $Z = 1$  – какие точки этой плоскости видны камерой?

## Вычисление луча из камеры

- Игнорируя ориентацию камеры, представим, что она находится в начале координат и её оси совпадают с осями координат (камера смотрит в направлении оси Z)
- Возьмём плоскость  $Z = 1$  – какие точки этой плоскости видны камерой?
- Точка  $(X, Y, 1)$  видна, если  $|X| < \tan\left(\frac{\text{fov}_X}{2}\right)$  и  $|Y| < \tan\left(\frac{\text{fov}_Y}{2}\right)$

## Вычисление луча из камеры

- Значит, нам нужно перевести диапазон пиксельных координат в координаты точек на плоскости  $Z = 1$ :
  - По X:  $[0, W] \rightarrow \left[ -\tan\left(\frac{\text{fov}_X}{2}\right), \tan\left(\frac{\text{fov}_X}{2}\right) \right]$
  - По Y:  $[0, H] \rightarrow \left[ -\tan\left(\frac{\text{fov}_Y}{2}\right), \tan\left(\frac{\text{fov}_Y}{2}\right) \right]$

## Вычисление луча из камеры

- Значит, нам нужно перевести диапазон пиксельных координат в координаты точек на плоскости  $Z = 1$ :
  - По X:  $[0, W] \rightarrow \left[ -\tan\left(\frac{\text{fov}_X}{2}\right), \tan\left(\frac{\text{fov}_X}{2}\right) \right]$
  - По Y:  $[0, H] \rightarrow \left[ -\tan\left(\frac{\text{fov}_Y}{2}\right), \tan\left(\frac{\text{fov}_Y}{2}\right) \right]$
- Это делается формулами
  - $P_X \mapsto \left(2 \frac{P_X}{W} - 1\right) \cdot \tan\left(\frac{\text{fov}_X}{2}\right)$
  - $P_Y \mapsto \left(2 \frac{P_Y}{H} - 1\right) \cdot \tan\left(\frac{\text{fov}_Y}{2}\right)$

## Вычисление луча из камеры

- Значит, нам нужно перевести диапазон пиксельных координат в координаты точек на плоскости  $Z = 1$ :
  - По X:  $[0, W] \rightarrow \left[ -\tan\left(\frac{\text{fov}_X}{2}\right), \tan\left(\frac{\text{fov}_X}{2}\right) \right]$
  - По Y:  $[0, H] \rightarrow \left[ -\tan\left(\frac{\text{fov}_Y}{2}\right), \tan\left(\frac{\text{fov}_Y}{2}\right) \right]$
- Это делается формулами
  - $P_X \mapsto \left(2\frac{P_X}{W} - 1\right) \cdot \tan\left(\frac{\text{fov}_X}{2}\right)$
  - $P_Y \mapsto \left(2\frac{P_Y}{H} - 1\right) \cdot \tan\left(\frac{\text{fov}_Y}{2}\right)$
- **N.B.:** Вместо точки  $(P_X, P_Y)$ , соответствующей углу пикселя, лучше брать его центр  $(P_X + 0.5, P_Y + 0.5)$
- **N.B.:** Обычно форматы изображений хранят пиксели сверху-вниз, и картинка получится перевёрнутой; проще всего это исправить, умножив формулу для Y на -1

## Вычисление луча из камеры

- Итак, по координатам пикселя  $(P_x, P_y)$  мы нашли вектор направления  $(X, Y, Z)$  с  $Z = 1$

## Вычисление луча из камеры

- Итак, по координатам пикселя  $(P_x, P_y)$  мы нашли вектор направления  $(X, Y, Z)$  с  $Z = 1$
- Как его перевести в систему координат камеры?

## Вычисление луча из камеры

- Итак, по координатам пикселя  $(P_x, P_y)$  мы нашли вектор направления  $(X, Y, Z)$  с  $Z = 1$
- Как его перевести в систему координат камеры? Взять линейную комбинацию осей камеры с этими коэффициентами:

$$D = X \cdot \text{right} + Y \cdot \text{up} + Z \cdot \text{forward} \quad (1)$$

## Вычисление луча из камеры

- Итак, по координатам пикселя  $(P_x, P_y)$  мы нашли вектор направления  $(X, Y, Z)$  с  $Z = 1$
- Как его перевести в систему координат камеры? Взять линейную комбинацию осей камеры с этими коэффициентами:

$$D = X \cdot \text{right} + Y \cdot \text{up} + Z \cdot \text{forward} \quad (1)$$

- Это и есть финальный вектор направления для нашего луча

## Вычисление луча из камеры

- Итак, по координатам пикселя  $(P_x, P_y)$  мы нашли вектор направления  $(X, Y, Z)$  с  $Z = 1$
- Как его перевести в систему координат камеры? Взять линейную комбинацию осей камеры с этими коэффициентами:

$$D = X \cdot \text{right} + Y \cdot \text{up} + Z \cdot \text{forward} \quad (1)$$

- Это и есть финальный вектор направления для нашего луча
- **N.B.:** Обычно вектор направления считают нормированным – это упрощает некоторые вычисления, но может усложнить другие; вы можете сделать так, как вам кажется удобнее

## Вычисление луча из камеры: aspect ratio

- Отношение ширины к высоте  $W/H$  изображения называется *aspect ratio*

## Вычисление луча из камеры: aspect ratio

- Отношение ширины к высоте  $W/H$  изображения называется *aspect ratio*
- Чтобы изображение получилось без искажений, нужно, чтобы это отношение совпадало с  $\tan\left(\frac{\text{fov}_x}{2}\right) / \tan\left(\frac{\text{fov}_y}{2}\right)$

## Вычисление луча из камеры: aspect ratio

- Отношение ширины к высоте  $W/H$  изображения называется *aspect ratio*
- Чтобы изображение получилось без искажений, нужно, чтобы это отношение совпадало с  $\tan\left(\frac{\text{fov}_x}{2}\right) / \tan\left(\frac{\text{fov}_y}{2}\right)$
- В формате сцены мы будем задавать только  $\text{fov}_x$ , а  $\text{fov}_y$  вычислять на основе остальных значений

## Вычисление пересечения луча и объектов

- Как вычислить пересечение луча и объекта сцены?

## Вычисление пересечения луча и объектов

- Как вычислить пересечение луча и объекта сцены? Зависит от *геометрии* объекта

## Вычисление пересечения луча и объектов

- Как вычислить пересечение луча и объекта сцены? Зависит от геометрии объекта
- Точки  $P = (X, Y, Z)$  луча удобно параметризовать параметром  $t \in [0, \infty)$  как  $P = O + t \cdot D$

## Вычисление пересечения луча и объектов

- Как вычислить пересечение луча и объекта сцены? Зависит от геометрии объекта
- Точки  $P = (X, Y, Z)$  луча удобно параметризовать параметром  $t \in [0, \infty)$  как  $P = O + t \cdot D$
- Позже мы будем использовать настоящие трёхмерные модели из треугольников, а пока ограничимся простыми геометрическими фигурами: плоскостью, эллипсоидом, параллелепипедом

# Вычисление пересечения луча и объектов

- Как вычислить пересечение луча и объекта сцены? Зависит от геометрии объекта
- Точки  $P = (X, Y, Z)$  луча удобно параметризовать параметром  $t \in [0, \infty)$  как  $P = O + t \cdot D$
- Позже мы будем использовать настоящие трёхмерные модели из треугольников, а пока ограничимся простыми геометрическими фигурами: плоскостью, эллипсоидом, параллелепипедом
- **N.B.:** Пока будем считать, что все объекты находятся в начале координат

## Вычисление пересечения луча и плоскости

- Плоскость удобно описывать вектором нормали и точкой, через которую проходит плоскость

## Вычисление пересечения луча и плоскости

- Плоскость удобно описывать вектором нормали и точкой, через которую проходит плоскость
- Мы договорились, что все объекты находятся в начале координат, так что нам достаточно вектора нормали  
 $N = (N_x, N_y, N_z)$

## Вычисление пересечения луча и плоскости

- Плоскость удобно описывать вектором нормали и точкой, через которую проходит плоскость
- Мы договорились, что все объекты находятся в начале координат, так что нам достаточно вектора нормали  $N = (N_x, N_y, N_z)$
- Уравнение такой плоскости:  $P \cdot N = X \cdot N_x + Y \cdot N_y + Z \cdot N_z = 0$

## Вычисление пересечения луча и плоскости

- Плоскость удобно описывать вектором нормали и точкой, через которую проходит плоскость
- Мы договорились, что все объекты находятся в начале координат, так что нам достаточно вектора нормали  $N = (N_x, N_y, N_z)$
- Уравнение такой плоскости:  $P \cdot N = X \cdot N_x + Y \cdot N_y + Z \cdot N_z = 0$
- Подставляем формулу для точки на луче, и получаем  $O \cdot N + t \cdot D \cdot N = 0$ , или  $t = -\frac{O \cdot N}{D \cdot N}$

## Вычисление пересечения луча и плоскости

- Плоскость удобно описывать вектором нормали и точкой, через которую проходит плоскость
- Мы договорились, что все объекты находятся в начале координат, так что нам достаточно вектора нормали  $N = (N_x, N_y, N_z)$
- Уравнение такой плоскости:  $P \cdot N = X \cdot N_x + Y \cdot N_y + Z \cdot N_z = 0$
- Подставляем формулу для точки на луче, и получаем  $O \cdot N + t \cdot D \cdot N = 0$ , или  $t = -\frac{O \cdot N}{D \cdot N}$
- Если  $t < 0$ , плоскость не видна в этом пикселе, иначе – видна

## Вычисление пересечения луча и сферы

- Сфера описывается радиусом  $R$  и уравнением

$$X^2 + Y^2 + Z^2 = P \cdot P = R^2 \quad (2)$$

## Вычисление пересечения луча и сферы

- Сфера описывается радиусом  $R$  и уравнением

$$X^2 + Y^2 + Z^2 = P \cdot P = R^2 \quad (2)$$

- Подставляем формулу точки луча, и получаем

$$O \cdot O + 2t(O \cdot D) + t^2(D \cdot D) = R^2 \quad (3)$$

## Вычисление пересечения луча и сферы

- Сфера описывается радиусом  $R$  и уравнением

$$X^2 + Y^2 + Z^2 = P \cdot P = R^2 \quad (2)$$

- Подставляем формулу точки луча, и получаем

$$O \cdot O + 2t(O \cdot D) + t^2(D \cdot D) = R^2 \quad (3)$$

- Это квадратное уравнение, у него может быть до двух решений:

## Вычисление пересечения луча и сферы

- Сфера описывается радиусом  $R$  и уравнением

$$X^2 + Y^2 + Z^2 = P \cdot P = R^2 \quad (2)$$

- Подставляем формулу точки луча, и получаем

$$O \cdot O + 2t(O \cdot D) + t^2(D \cdot D) = R^2 \quad (3)$$

- Это квадратное уравнение, у него может быть до двух решений:
  - Если решений нет, луч не пересекает сферу
  - Если решение одно, луч касается сферы
  - Если решений два, луч пересекает сферу в двух точках и проходит через её внутренность

## Вычисление пересечения луча и сферы

- Сфера описывается радиусом  $R$  и уравнением

$$X^2 + Y^2 + Z^2 = P \cdot P = R^2 \quad (2)$$

- Подставляем формулу точки луча, и получаем

$$O \cdot O + 2t(O \cdot D) + t^2(D \cdot D) = R^2 \quad (3)$$

- Это квадратное уравнение, у него может быть до двух решений:
  - Если решений нет, луч не пересекает сферу
  - Если решение одно, луч касается сферы
  - Если решений два, луч пересекает сферу в двух точках и проходит через её внутренность
- Нам нет нужды различать случаи одного и двух решений – будем считать, что их всегда два

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения
- Могут произойти три вещи:

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения
- Могут произойти три вещи:
  - $0 < t_1$  – камера находится снаружи сферы,  $t_1$  – ближайшее пересечение

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения
- Могут произойти три вещи:
  - $0 < t_1$  – камера находится снаружи сферы,  $t_1$  – ближайшее пересечение
  - $t_1 < 0 < t_2$  – камера находится внутри сферы,  $t_1$  – точка сзади камеры,  $t_2$  – ближайшее видимое пересечение

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения
- Могут произойти три вещи:
  - $0 < t_1$  – камера находится снаружи сферы,  $t_1$  – ближайшее пересечение
  - $t_1 < 0 < t_2$  – камера находится внутри сферы,  $t_1$  – точка сзади камеры,  $t_2$  – ближайшее видимое пересечение
  - $t_2 < 0$  – камера находится снаружи сферы,  $t_2$  – точка сзади камеры, нет видимого пересечения

## Вычисление пересечения луча и сферы

- Пусть  $t_1 < t_2$  – два решения квадратного уравнения
- Могут произойти три вещи:
  - $0 < t_1$  – камера находится снаружи сферы,  $t_1$  – ближайшее пересечение
  - $t_1 < 0 < t_2$  – камера находится внутри сферы,  $t_1$  – точка сзади камеры,  $t_2$  – ближайшее видимое пересечение
  - $t_2 < 0$  – камера находится снаружи сферы,  $t_2$  – точка сзади камеры, нет видимого пересечения
- То есть среди значений  $t_1, t_2$  нас интересует минимальное, большее нуля; если таких нет, то сфера не видна этим лучом

## Вычисление пересечения луча и эллипсоида

- Эллипсоид описывается тремя радиусами  $R = (R_X, R_Y, R_Z)$  и уравнением

$$\frac{x^2}{R_X^2} + \frac{y^2}{R_Y^2} + \frac{z^2}{R_Z^2} = \frac{P}{R} \cdot \frac{P}{R} = 1 \quad (4)$$

## Вычисление пересечения луча и эллипсоида

- Эллипсоид описывается тремя радиусами  $R = (R_X, R_Y, R_Z)$  и уравнением

$$\frac{x^2}{R_X^2} + \frac{y^2}{R_Y^2} + \frac{z^2}{R_Z^2} = \frac{P}{R} \cdot \frac{P}{R} = 1 \quad (4)$$

- Здесь,  $\frac{P}{R}$  – *покомпонентное* деление векторов

## Вычисление пересечения луча и эллипсоида

- Эллипсоид описывается тремя радиусами  $R = (R_X, R_Y, R_Z)$  и уравнением

$$\frac{x^2}{R_X^2} + \frac{y^2}{R_Y^2} + \frac{z^2}{R_Z^2} = \frac{P}{R} \cdot \frac{P}{R} = 1 \quad (4)$$

- Здесь,  $\frac{P}{R}$  – *покомпонентное* деление векторов
- Подставляем формулу точки луча, и получаем

$$\frac{O}{R} \cdot \frac{O}{R} + 2t \left( \frac{O}{R} \cdot \frac{D}{R} \right) + t^2 \left( \frac{D}{R} \cdot \frac{D}{R} \right) = 1 \quad (5)$$

## Вычисление пересечения луча и эллипсоида

- Эллипсоид описывается тремя радиусами  $R = (R_X, R_Y, R_Z)$  и уравнением

$$\frac{x^2}{R_X^2} + \frac{y^2}{R_Y^2} + \frac{z^2}{R_Z^2} = \frac{P}{R} \cdot \frac{P}{R} = 1 \quad (4)$$

- Здесь,  $\frac{P}{R}$  – *покомпонентное* деление векторов
- Подставляем формулу точки луча, и получаем

$$\frac{O}{R} \cdot \frac{O}{R} + 2t \left( \frac{O}{R} \cdot \frac{D}{R} \right) + t^2 \left( \frac{D}{R} \cdot \frac{D}{R} \right) = 1 \quad (5)$$

- Это квадратное уравнение, дальше всё аналогично ситуации со сферой

## Вычисление пересечения луча и параллелепипеда

- Считая, что центр параллелепипеда находится в центре координат, удобно его описать размерами по трём осям  $S = (S_x, S_y, S_z)$  и уравнениями внутренности

$$|X| < S_x \quad (6)$$

$$|Y| < S_y \quad (7)$$

$$|Z| < S_z \quad (8)$$

## Вычисление пересечения луча и параллелепипеда

- Считая, что центр параллелепипеда находится в центре координат, удобно его описать размерами по трём осям  $S = (S_x, S_y, S_z)$  и уравнениями внутренности

$$|X| < S_x \quad (6)$$

$$|Y| < S_y \quad (7)$$

$$|Z| < S_z \quad (8)$$

- Удобно представить параллелепипед как пересечение трёх бесконечных множеств  $|X| < S_x$ ,  $|Y| < S_y$  и  $|Z| < S_z$

## Вычисление пересечения луча и параллелепипеда

- Считая, что центр параллелепипеда находится в центре координат, удобно его описать размерами по трём осям  $S = (S_x, S_y, S_z)$  и уравнениями внутренности

$$|X| < S_x \quad (6)$$

$$|Y| < S_y \quad (7)$$

$$|Z| < S_z \quad (8)$$

- Удобно представить параллелепипед как пересечение трёх бесконечных множеств  $|X| < S_x$ ,  $|Y| < S_y$  и  $|Z| < S_z$
- Подставляем формулу точки луча в уравнение границы по X:  $X = \pm S_x$  и получаем

$$(O + t \cdot D)_x = \pm S_x \implies t = \frac{\pm S_x - O_x}{D_x} \quad (9)$$

## Вычисление пересечения луча и параллелепипеда

- Пусть  $t_{1,x} < t_{2,x}$  – значения, полученные по формуле с предыдущего слайда

## Вычисление пересечения луча и параллелепипеда

- Пусть  $t_{1,x} < t_{2,x}$  – значения, полученные по формуле с предыдущего слайда
- Аналогично  $t_{1,y} < t_{2,y}$  и  $t_{1,z} < t_{2,z}$

## Вычисление пересечения луча и параллелепипеда

- Пусть  $t_{1,x} < t_{2,x}$  – значения, полученные по формуле с предыдущего слайда
- Аналогично  $t_{1,y} < t_{2,y}$  и  $t_{1,z} < t_{2,z}$
- Мы получили три диапазона значений  $t$ , нас интересует их пересечение:

$$t_1 = \max(t_{1,x}, t_{1,y}, t_{1,z}) \quad (10)$$

$$t_2 = \min(t_{2,x}, t_{2,y}, t_{2,z}) \quad (11)$$

## Вычисление пересечения луча и параллелепипеда

- Пусть  $t_{1,x} < t_{2,x}$  – значения, полученные по формуле с предыдущего слайда
- Аналогично  $t_{1,y} < t_{2,y}$  и  $t_{1,z} < t_{2,z}$
- Мы получили три диапазона значений  $t$ , нас интересует их пересечение:

$$t_1 = \max(t_{1,x}, t_{1,y}, t_{1,z}) \quad (10)$$

$$t_2 = \min(t_{2,x}, t_{2,y}, t_{2,z}) \quad (11)$$

- Если  $t_1 > t_2$ , то пересечения нет

## Вычисление пересечения луча и параллелепипеда

- Пусть  $t_{1,x} < t_{2,x}$  – значения, полученные по формуле с предыдущего слайда
- Аналогично  $t_{1,y} < t_{2,y}$  и  $t_{1,z} < t_{2,z}$
- Мы получили три диапазона значений  $t$ , нас интересует их пересечение:

$$t_1 = \max(t_{1,x}, t_{1,y}, t_{1,z}) \quad (10)$$

$$t_2 = \min(t_{2,x}, t_{2,y}, t_{2,z}) \quad (11)$$

- Если  $t_1 > t_2$ , то пересечения нет
- Иначе – аналогично сфере, сравниваем значения с нулём

## Вычисление пересечения луча и сдвинутого объекта

- Что, если объект находится не в начале координат, а в точке  $T$ ?

## Вычисление пересечения луча и сдвинутого объекта

- Что, если объект находится не в начале координат, а в точке  $T$ ?
- Переместим на вектор  $-T$  и объект, и луч, т.е вычтем  $T$  из точки начала луча (объект переместим только мысленно)

## Вычисление пересечения луча и сдвинутого объекта

- Что, если объект находится не в начале координат, а в точке  $T$ ?
- Переместим на вектор  $-T$  и объект, и луч, т.е вычтем  $T$  из точки начала луча (объект переместим только мысленно)
- Теперь объект в начале координат  $\Rightarrow$  используем описанные ранее алгоритмы

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут?

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут?
- Есть разные способы описания вращений: матрицы, углы Эйлера, кватернионы

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут?
- Есть разные способы описания вращений: матрицы, углы Эйлера, кватернионы
- Мы будем использовать кватернионы

## Кватернионы $\mathbb{H}$ : TL;DR

- $\mathbb{H}$  – четырёхмерная алгебра над вещественными числами  $\mathbb{R}$ , т.е. описываются четырьмя координатами  $(X, Y, Z, W)$

## Кватернионы $\mathbb{H}$ : TL;DR

- $\mathbb{H}$  – четырёхмерная алгебра над вещественными числами  $\mathbb{R}$ , т.е. описываются четырьмя координатами  $(X, Y, Z, W)$
- Можно складывать, умножать, делить (умножение не коммутативно)

## Кватернионы $\mathbb{H}$ : TL;DR

- $\mathbb{H}$  – четырёхмерная алгебра над вещественными числами  $\mathbb{R}$ , т.е. описываются четырьмя координатами  $(X, Y, Z, W)$
- Можно складывать, умножать, делить (умножение не коммутативно)
- Любое вращение трёхмерного пространства описывается двумя противоположными ( $Q_1 = -Q_2$ ) единичными  $(X^2 + Y^2 + Z^2 + W^2 = 1)$  кватернионами

## Кватернионы $\mathbb{H}$ : TL;DR

- Кватернион  $(X, Y, Z, W)$  удобно разбить на скалярную  $W$  и векторную  $V = (X, Y, Z)$  части

## Кватернионы $\mathbb{H}$ : TL;DR

- Кватернион  $(X, Y, Z, W)$  удобно разбить на скалярную  $W$  и векторную  $V = (X, Y, Z)$  части
- Умножение кватернионов в такой форме:

$$(V_1, W_1) \cdot (V_2, W_2) = (W_1 \cdot V_2 + W_2 \cdot V_1 + V_1 \times V_2, W_1 \cdot W_2 - V_1 \cdot V_2) \quad (12)$$

## Кватернионы $\mathbb{H}$ : TL;DR

- Кватернион  $(X, Y, Z, W)$  удобно разбить на скалярную  $W$  и векторную  $V = (X, Y, Z)$  части
- Умножение кватернионов в такой форме:

$$(V_1, W_1) \cdot (V_2, W_2) = (W_1 \cdot V_2 + W_2 \cdot V_1 + V_1 \times V_2, W_1 \cdot W_2 - V_1 \cdot V_2) \quad (12)$$

- Сопряжённый кватернион:

$$\overline{(V, W)} = (-V, W) \quad (13)$$

## Кватернионы и вращения

- Любой единичный кватернион  $(V, W)$  определяет вращение вектора  $X$  по формуле

$$X \mapsto (V, W) \cdot (X, 0) \cdot (-V, W) \quad (14)$$

# Кватернионы и вращения

- Любой единичный кватернион  $(V, W)$  определяет вращение вектора  $X$  по формуле

$$X \mapsto (V, W) \cdot (X, 0) \cdot (-V, W) \quad (14)$$

- Вращение на угол  $\theta$  вокруг вектора  $V$  задаёт кватернион

$$\left( \sin \frac{\theta}{2} \cdot V, \cos \frac{\theta}{2} \right) \quad (15)$$

# Кватернионы и вращения

- Любой единичный кватернион  $(V, W)$  определяет вращение вектора  $X$  по формуле

$$X \mapsto (V, W) \cdot (X, 0) \cdot (-V, W) \quad (14)$$

- Вращение на угол  $\theta$  вокруг вектора  $V$  задаёт кватернион

$$\left( \sin \frac{\theta}{2} \cdot V, \cos \frac{\theta}{2} \right) \quad (15)$$

- Сопряжённый кватернион описывает обратное вращение

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут на кватернион  $Q$ ?

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут на кватернион  $Q$ ?
- Повернём и объект, и вектор направления луча на сопряжённый кватернион  $\bar{Q}$  (объект – только мысленно)

## Вычисление пересечения луча и повёрнутого объекта

- Что, если объект не параллелен осям координат, а повернут на кватернион  $Q$ ?
- Повернём и объект, и вектор направления луча на сопряжённый кватернион  $\bar{Q}$  (объект – только мысленно)
- Теперь объект параллелен осям координат  $\Rightarrow$  используем описанные ранее алгоритмы

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта
- Нам нужно обратное преобразование:  $X = R_Q^{-1} \cdot (Y - T)$

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта
- Нам нужно обратное преобразование:  $X = R_Q^{-1} \cdot (Y - T)$
- Тогда алгоритм для преобразования луча такой:

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта
- Нам нужно обратное преобразование:  $X = R_Q^{-1} \cdot (Y - T)$
- Тогда алгоритм для преобразования луча такой:
  - Вычтем  $T$  из точки начала луча, и повернёт результат на сопряжённый кватернион  $\bar{Q}$

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта
- Нам нужно обратное преобразование:  $X = R_Q^{-1} \cdot (Y - T)$
- Тогда алгоритм для преобразования луча такой:
  - Вычтем  $T$  из точки начала луча, и повернёт результат на сопряжённый кватернион  $\bar{Q}$
  - Повернём вектор направления луча на сопряжённый кватернион  $\bar{Q}$

## Вычисление пересечения луча и произвольного объекта

- Что, если объект и повёрнут, и сдвинут?
- Преобразование из ‘стандартного’ объекта в повёрнутый и сдвинутый осуществляется как  $Y = T + R_Q \cdot X$ , где  $R_Q$  – оператор вращения на кватернион  $Q$ , а  $X$  – точка объекта
- Нам нужно обратное преобразование:  $X = R_Q^{-1} \cdot (Y - T)$
- Тогда алгоритм для преобразования луча такой:
  - Вычтем  $T$  из точки начала луча, и повернём результат на сопряжённый кватернион  $\bar{Q}$
  - Повернём вектор направления луча на сопряжённый кватернион  $\bar{Q}$
- Используем описанные ранее алгоритмы с преобразованным лучом

## Вычисление пересечения луча и сцены

- Проходим по всем объектам сцены, находим пересечение луча с ними

## Вычисление пересечения луча и сцены

- Проходим по всем объектам сцены, находим пересечение луча с ними
- Из всех объектов, пересекших луч, находим ближайшее к камере пересечение, т.е. с минимальным значением  $t$

## Вычисление пересечения луча и сцены

- Проходим по всем объектам сцены, находим пересечение луча с ними
- Из всех объектов, пересекших луч, находим ближайшее к камере пересечение, т.е. с минимальным значением  $t$
- Если было хотя бы одно пересечение, записываем в пиксель цвет ближайшего объекта

## Вычисление пересечения луча и сцены

- Проходим по всем объектам сцены, находим пересечение луча с ними
- Из всех объектов, пересекших луч, находим ближайшее к камере пересечение, т.е. с минимальным значением  $t$
- Если было хотя бы одно пересечение, записываем в пиксель цвет ближайшего объекта
- Если пересечений не было, записываем в пиксель цвет фона

# Сохранение изображения

- Большинство форматов изображений довольно сложны для того, чтобы вручную их записывать, – для этого нужны сторонние библиотеки

# Сохранение изображения

- Большинство форматов изображений довольно сложны для того, чтобы вручную их записывать, – для этого нужны сторонние библиотеки
- Есть формат NetPBM (.pbm/.pgm/.ppm), ориентированный на простоту чтения и записи

# Сохранение изображения

- Большинство форматов изображений довольно сложны для того, чтобы вручную их записывать, – для этого нужны сторонние библиотеки
- Есть формат NetPBM (.pbm/.pgm/.ppm), ориентированный на простоту чтения и записи
- Его поддерживают многие редакторы, и большинство просмотрщиков изображений

# Сохранение изображения

- Большинство форматов изображений довольно сложны для того, чтобы вручную их записывать, – для этого нужны сторонние библиотеки
- Есть формат NetPBM (.pbm/.pgm/.ppm), ориентированный на простоту чтения и записи
- Его поддерживают многие редакторы, и большинство просмотрщиков изображений
- Мы будем использовать бинарную версию формата PPM (P6, его описание есть в слайдах практики)

## Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей

## Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей
- Удобнее всего хранить пиксели как непрерывный массив байт (напр. `vector<uint8_t>` в C++), последовательно хранящий строки изображения

## Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей
- Удобнее всего хранить пиксели как непрерывный массив байт (напр. `vector<uint8_t>` в C++), последовательно хранящий строки изображения
- Пиксель – три байта: его **красный**, **зелёный** и **синий** каналы, соответственно (в диапазоне [0..255])

## Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей
- Удобнее всего хранить пиксели как непрерывный массив байт (напр. `vector<uint8_t>` в C++), последовательно хранящий строки изображения
- Пиксель – три байта: его **красный**, **зелёный** и **синий** каналы, соответственно (в диапазоне [0..255])
- Для доступа к i-ой компоненте пикселя (x,y) в непрерывном массиве можно использовать  
`pixels[(y * width + x) * 3 + i]`

# Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей
- Удобнее всего хранить пиксели как непрерывный массив байт (напр. `vector<uint8_t>` в C++), последовательно хранящий строки изображения
- Пиксель – три байта: его **красный**, **зелёный** и **синий** каналы, соответственно (в диапазоне [0..255])
- Для доступа к  $i$ -ой компоненте пикселя  $(x,y)$  в непрерывном массиве можно использовать  
`pixels[(y * width + x) * 3 + i]`
- Мы будем работать с цветами как с floating-point числами, как правило, в диапазоне [0..1]

# Сохранение изображения

- Для записи изображения в файл нам потребуется массив его пикселей
- Удобнее всего хранить пиксели как непрерывный массив байт (напр. `vector<uint8_t>` в C++), последовательно хранящий строки изображения
- Пиксель – три байта: его **красный**, **зелёный** и **синий** каналы, соответственно (в диапазоне [0..255])
- Для доступа к  $i$ -ой компоненте пикселя  $(x,y)$  в непрерывном массиве можно использовать  
`pixels[(y * width + x) * 3 + i]`
- Мы будем работать с цветами как с floating-point числами, как правило, в диапазоне [0..1]
- После того, как цвет пикселя вычислен, его нужно будет сконвертировать в целое число [0..255]

## Литература, ссылки

- Серия книжек *Raytracing In One Weekend* – очень хороший учебник начального уровня
- *Computer Graphics from Scratch* – книжка начального уровня
- *Physically Based Rendering: From Theory To Implementation* – книжка продвинутого уровня
- [realtimerendering.com/raytracing.html](http://realtimerendering.com/raytracing.html) – сборник ресурсов про рейтрейсинг