

# Компьютерная графика

## Лекция 1: Введение в курс

2021

# 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)



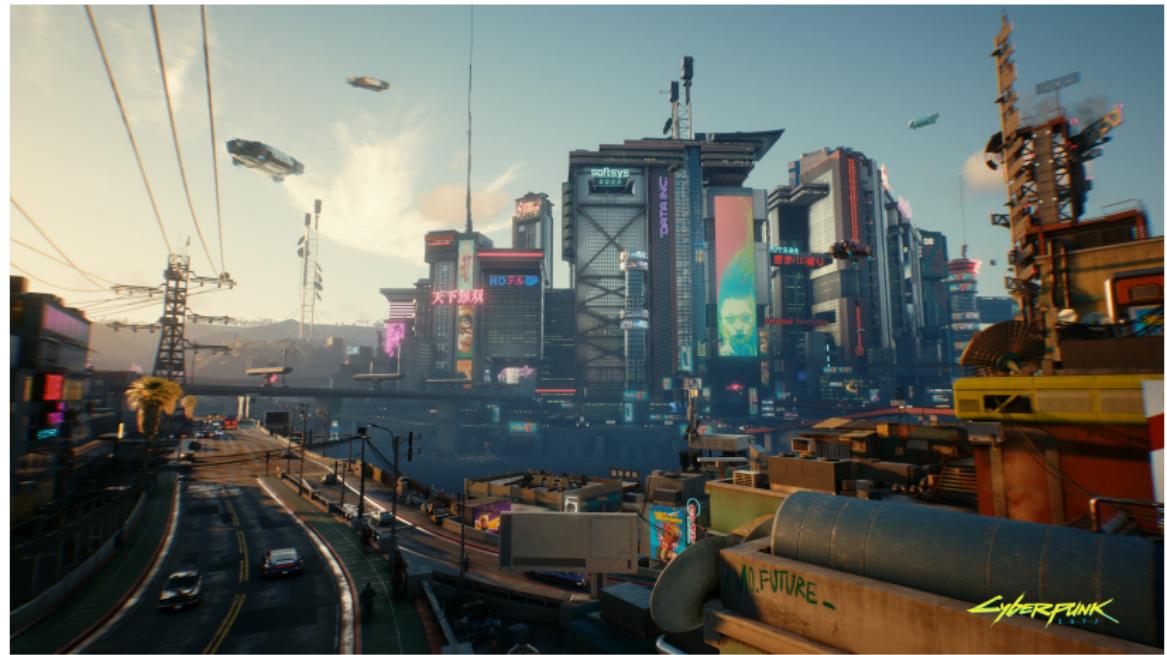
## Civilization V (2010)



# The Witcher 3: Wild Hunt (2015)



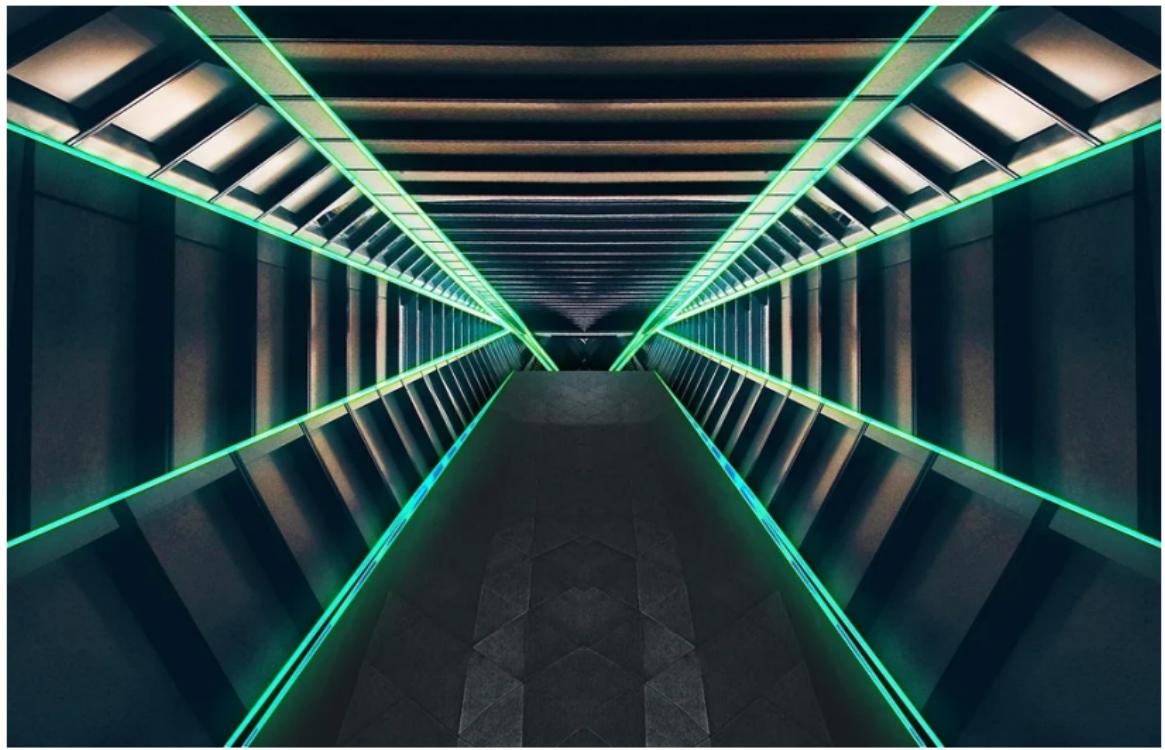
# Cyberpunk 2077 (2020)



# 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 <input type="text" value="mm/dd/yyyy"/>	Label title <input type="text" value="Placeholder text"/>	Label title <input type="text" value="Placeholder text"/>																																																	
Disabled button	Hover <input type="text" value="mm/dd/yyyy"/>	Hover <input type="text" value="Placeholder text"/>	Hover <input type="text" value="Placeholder text"/>																																																	
Toggle button	Disabled <input type="text" value="mm/dd/yyyy"/>	Disabled <input type="text" value="Placeholder text"/>	Disabled <input type="text" value="Placeholder text"/>																																																	
Checkbox	Disabled <input type="checkbox"/> Unchecked <input checked="" type="checkbox"/> Checked <input type="checkbox"/> Third state <input checked="" type="checkbox"/> Disabled	February 2018 <table border="1"><thead><tr><th>Sun</th><th>Mon</th><th>Tue</th><th>Wed</th><th>Thu</th><th>Fri</th><th>Sat</th></tr></thead><tbody><tr><td>31</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td></tr><tr><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td></tr><tr><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td></tr><tr><td>28</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td></tr></tbody></table>	Sun	Mon	Tue	Wed	Thu	Fri	Sat	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11	12	13	Typing <input type="text" value="This is text."/> Password <input type="password" value="*****"/>
Sun	Mon	Tue	Wed	Thu	Fri	Sat																																														
31	1	2	3	4	5	6																																														
7	8	9	10	11	12	13																																														
14	15	16	17	18	19	20																																														
21	22	23	24	25	26	27																																														
28	1	2	3	4	5	6																																														
7	8	9	10	11	12	13																																														
Radio button	Unchecked <input type="radio"/> Checked <input checked="" type="radio"/> Disabled <input type="radio"/>	Microsoft Windows Office	Toggle switch Off On Disabled Off Disabled On																																																	

# Europa Universalis 4

The screenshot shows the Brandenburg interface in Europa Universalis 4. The top bar displays resources: Gold (67), Food (13,023), Silver (0), Coal (0), Oil (0), Wood (7), and Population (100). It also shows diplomatic relations with Poland (0/2), France (0/0), and Russia (2/2, 1/1). The date is 1444.

**Brandenburg** (Red Dragon)

**Diplomatic Relations:**

- Free Christian von Anhalt ...
- Free Clemens von Quern ...

**Merchants:**

- Saxony 0.43 → to Lübeck
- Rheinland 0.37 → to Lübeck

**Armies:** 8,000 Kurfürstliche Garde

**Demographics:**

- Tax 0.26
- Production 0.10
- Total 0.36
- Unrest 0.0
- Autonomy 5.0%

**Cores & Claims:** Bohemia (+22)

**Culture:** Saxon

**Religion:** Catholic

**Diplomacy:**

**Military:**

- Mapower 261
- Supply Limit 13
- Sailors 0 (11%)
- Garrison: 0
- No Objective

**Trade:**

- Trade Power 1.6
- Trade Value 1.23
- Goods Produced 0.61

**Buildings:** 10

**Estates:** No Estate

**Province Values:**

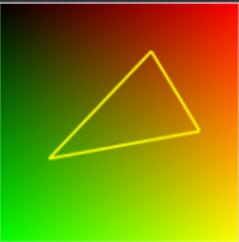
- Food 2 (0.00)
- Wood 0.00 (0.66)

Bottom status bar: EN, 9:30 PM, 28-May-16, system icons.

# Dear ImGui

▼ Example: Custom rendering

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



E028E4: 00 00 F0 44 00 40 7E 44 ...D.@@D  
E028EC: 06 33 88 3C 00 00 A0 40 ...<...@  
E028F4: 18 BADE 00 08 BA DE 00 .....@...  
E028FC: 9A 99 99 3E 00 00 C0 40 .....@...  
E02984: 00 00 C0 40 02 01 00 00 .....@...  
E0298C: 07 01 00 00 06 01 00 00 .....@...  
E02914: 89 01 00 00 08 01 00 00 .....@...  
E0291C: 8A 01 00 00 08 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 MailX\_Console  
E0294C: 59 00 00 00 5A 00 00 00 Y...Z..  
E02954: 00 00 88 3E CD CC 4C 3D #>@!<@!& implements a console with basic coloring,  
E0295C: 00 00 00 00 A4 28 E0 @!<@!& history. A more elaborate implementation may  
want to store entries along with extra data such as timestamp,  
8 rows Range E028E4..E03C93 Iter: etc.

▼ 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  
MuObject 13

Revert Save

MyObject: 0  
Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor  
incididunt ut labore et dolore  
magna aliqua.

Enter 'HELP' for help, press TAB to use text completion.  
Add Dummy Text Add Dummy Error Clear  
Filter ("incl,-exc1") ("error")

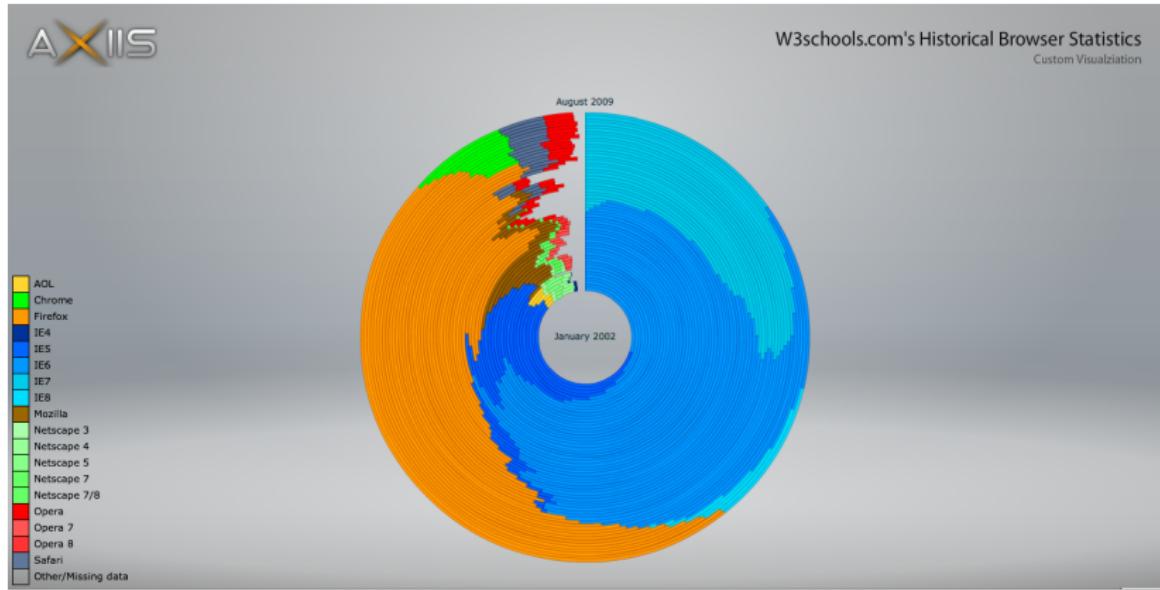
0 some text  
some more text  
display very important message here!  
[error] something went wrong  
Possible matches:  
- HELP  
- HISTORY  
# HELP  
Commands:  
- HELP  
- HISTORY  
- CLEAR  
- CLASSIFY  
# hello, imgui world!  
Unknown command: 'hello, imgui world!'

hello, imgui world! Input

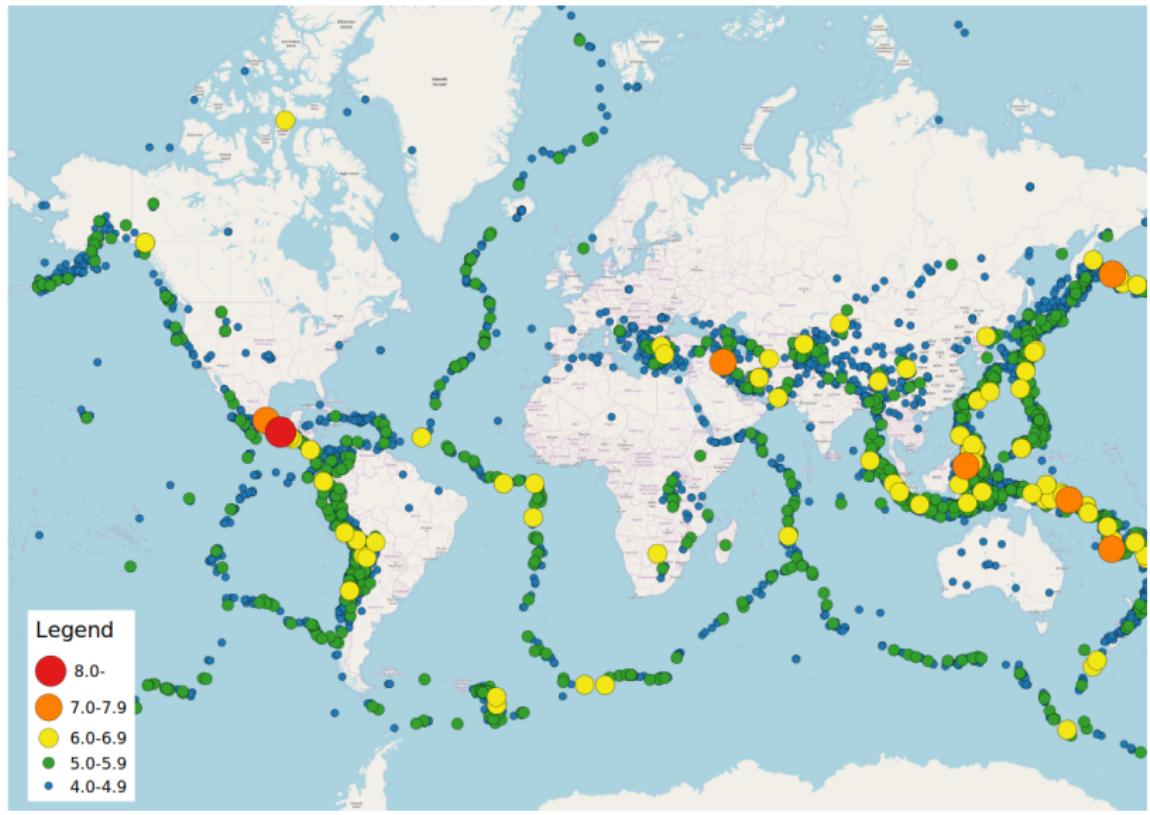
# Что такое компьютерная графика?

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

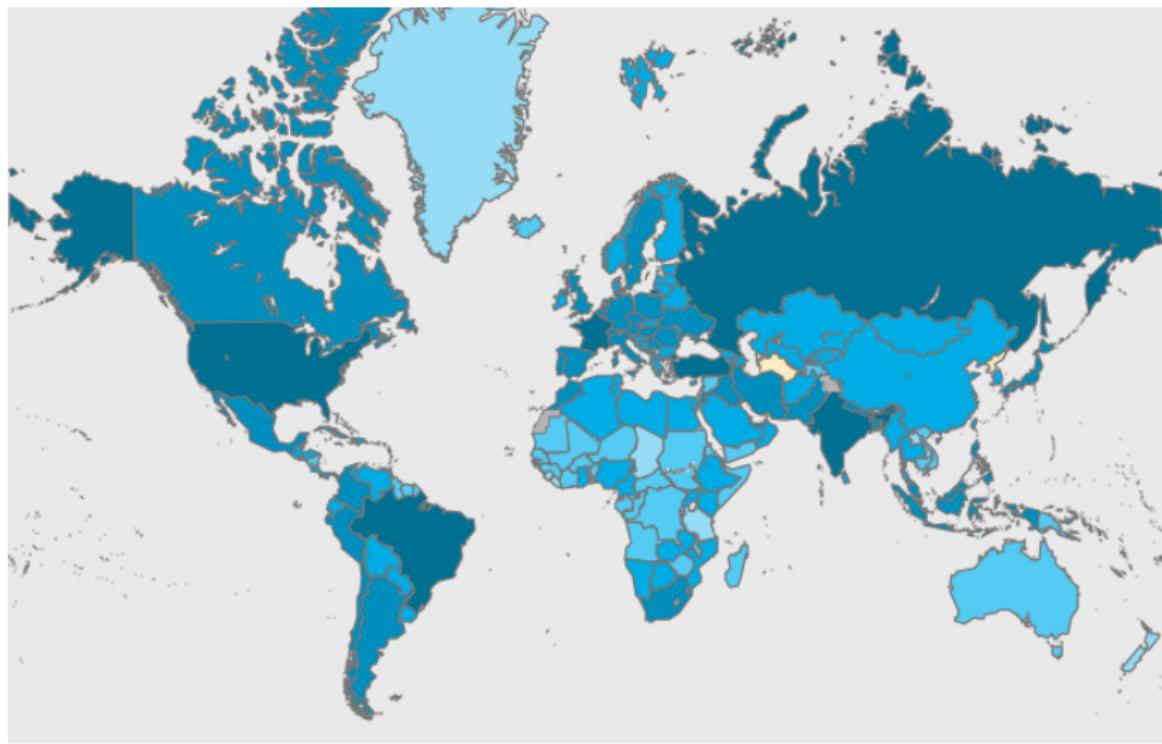
# Популярность браузеров в 2002-2009



# Карта землетрясений



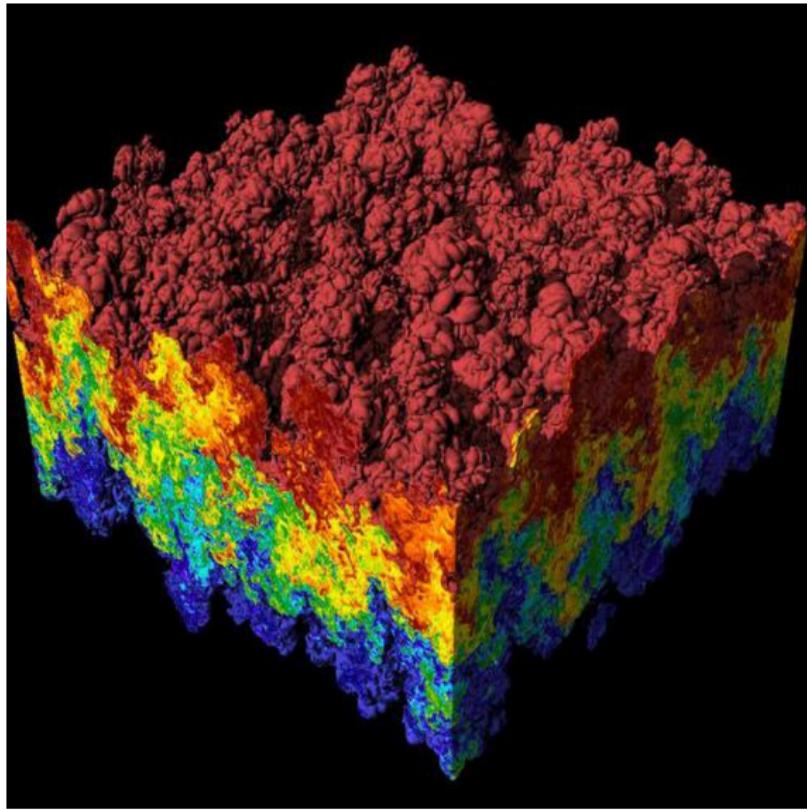
# Количество случаев заражения COVID-19



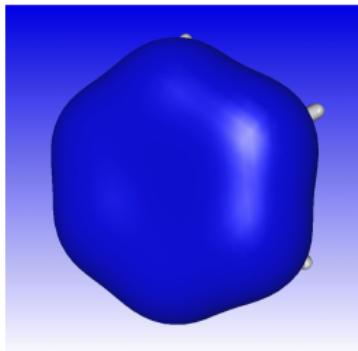
# Что такое компьютерная графика?

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

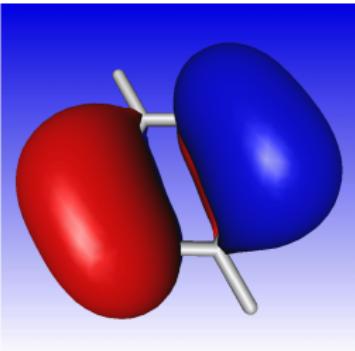
# Неустойчивость Рэлея — Тейлора



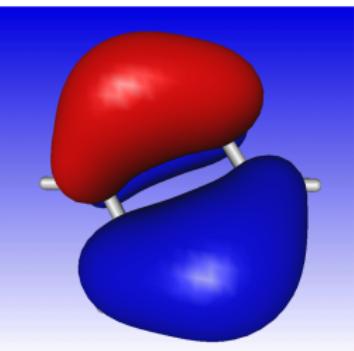
# Молекулярные орбитали бензола



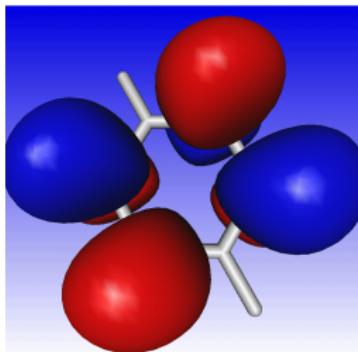
16



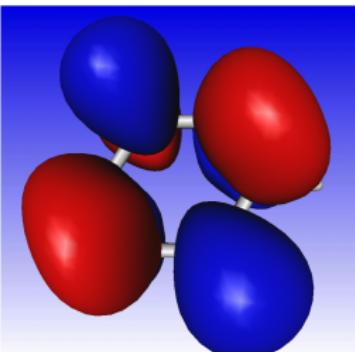
20



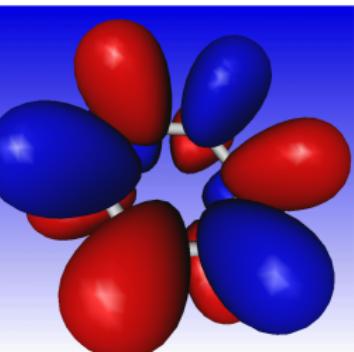
21



22

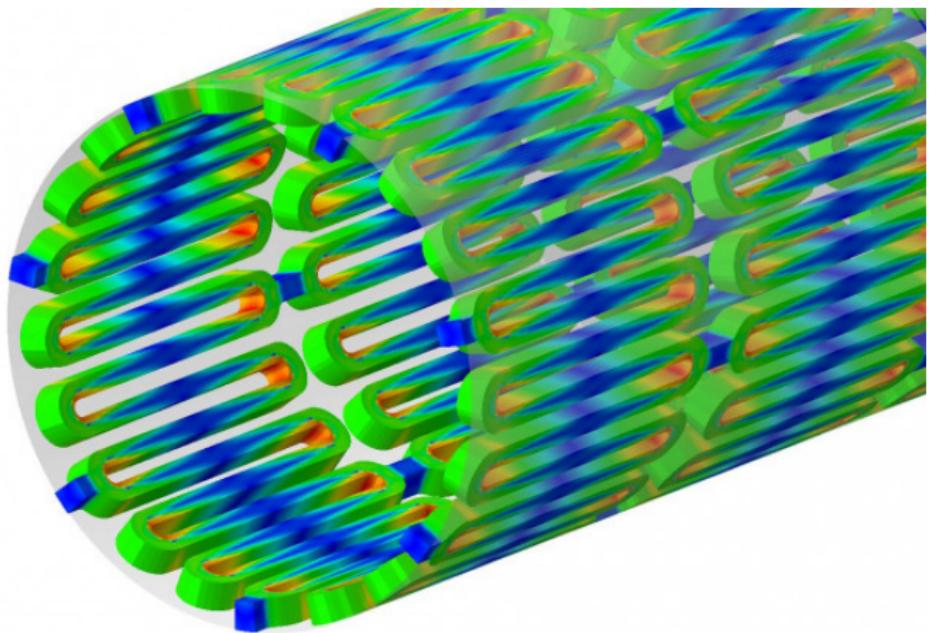


23



30

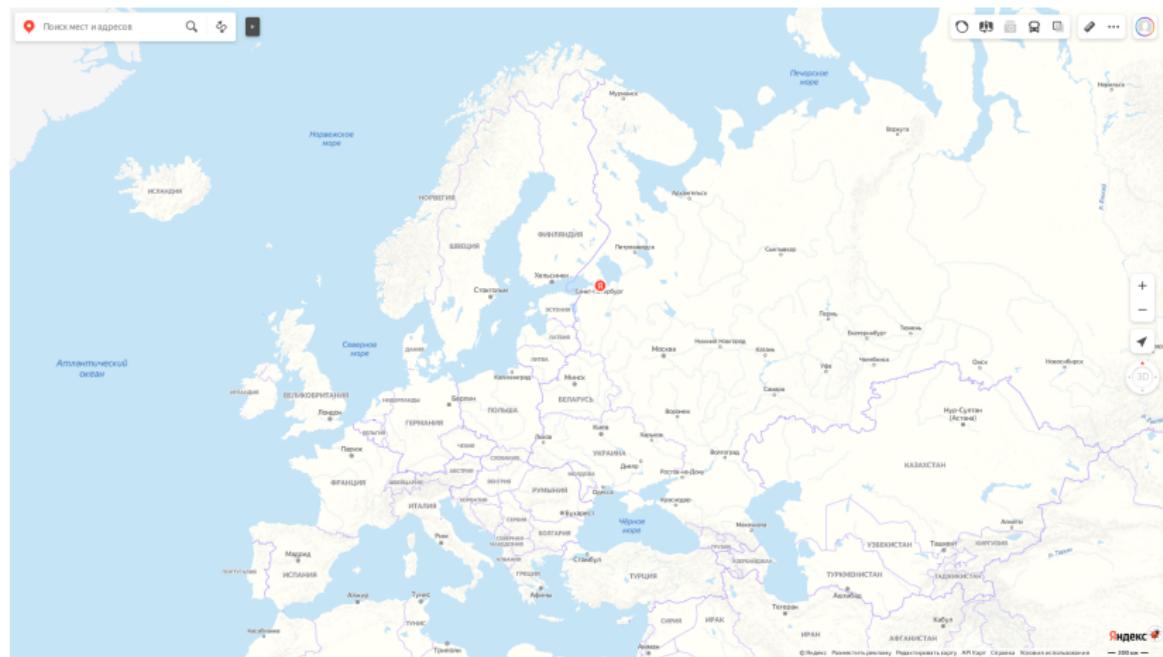
# Симуляция напряжений в стенке методом конечных элементов



# Что такое компьютерная графика?

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

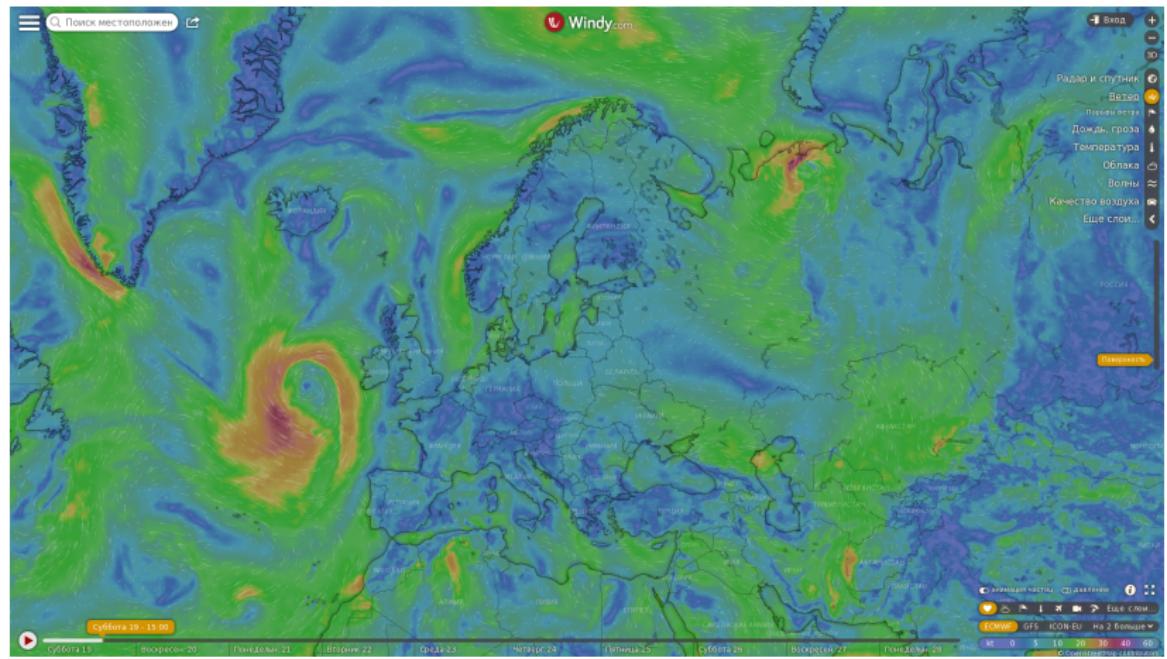
# Схематическая карта



## 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

# Грубая и неточная классификация

- ▶ 2D / 2.5D / 3D

# 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

# Как использовать GPU?

GPU - Graphics Processing Unit

# Как использовать GPU? Графические API:

GPU - Graphics Processing Unit

# Как использовать GPU? Графические API:

GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)
  - ▶ DirectX 12 (Microsoft, 2015)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)
  - ▶ DirectX 12 (Microsoft, 2015)
- ▶ Metal (Apple, 2014)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)
  - ▶ DirectX 12 (Microsoft, 2015)
- ▶ Metal (Apple, 2014)
- ▶ Vulkan (Khronos Group, 2018)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)
  - ▶ DirectX 12 (Microsoft, 2015)
- ▶ Metal (Apple, 2014)
- ▶ Vulkan (Khronos Group, 2018)

# Как использовать GPU? Графические API:

## GPU - Graphics Processing Unit

- ▶ Вендор-специфичные API (1980е - 1990е)
- ▶ OpenGL (Silicon Graphics, 1992)
  - ▶ OpenGL 3.3 (Khronos Group, 2010)
- ▶ DirectX (Microsoft, 1995)
  - ▶ DirectX 12 (Microsoft, 2015)
- ▶ Metal (Apple, 2014)
- ▶ Vulkan (Khronos Group, 2018)

# Как использовать GPU? API общего назначения (GPGPU):

GPGPU - General-Purpose Graphics Processing Unit

# Как использовать GPU? API общего назначения (GPGPU):

GPGPU - General-Purpose Graphics Processing Unit

- ▶ CUDA (Nvidia, 2007)

# Как использовать GPU? API общего назначения (GPGPU):

GPGPU - General-Purpose Graphics Processing Unit

- ▶ CUDA (Nvidia, 2007)
- ▶ DirectX 11 DirectCompute (Microsoft, 2008)

# Как использовать GPU? API общего назначения (GPGPU):

GPGPU - General-Purpose Graphics Processing Unit

- ▶ CUDA (Nvidia, 2007)
- ▶ DirectX 11 DirectCompute (Microsoft, 2008)
- ▶ OpenCL (Khronos Group, 2009)

# Как использовать GPU? API общего назначения (GPGPU):

GPGPU - General-Purpose Graphics Processing Unit

- ▶ CUDA (Nvidia, 2007)
- ▶ DirectX 11 DirectCompute (Microsoft, 2008)
- ▶ OpenCL (Khronos Group, 2009)
- ▶ OpenGL 4.3 Compute Shaders (Khronos Group, 2012)

# Почему OpenGL 3.3?

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web
- ▶ Поддерживает всё, что нам нужно

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web
- ▶ Поддерживает всё, что нам нужно
- ▶ +/- Кроссплатформенность

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web
- ▶ Поддерживает всё, что нам нужно
- ▶ +/- Кроссплатформенность
- ▶ Низкий порог входления

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web
- ▶ Поддерживает всё, что нам нужно
- ▶ +/- Кроссплатформенность
- ▶ Низкий порог входления
- ▶ Достаточно старый API
  - ▶ Много вспомогательных библиотек
  - ▶ Известны best practices
  - ▶ Известны все грабли

# Почему OpenGL 3.3?

- ▶ Широкая поддержка: интегрированные GPU, встраиваемые устройства, телефоны, web
- ▶ Поддерживает всё, что нам нужно
- ▶ +/- Кроссплатформенность
- ▶ Низкий порог входления
- ▶ Достаточно старый API
  - ▶ Много вспомогательных библиотек
  - ▶ Известны best practices
  - ▶ Известны все грабли (их много)

# История графических API: OpenGL 1.0

```
for o in scene.objects:  
    glBegin(GL_TRIANGLES)  
    for t in o.triangles:  
        for v in t.vertices:  
            glColor3f(v.color)  
            glNormal3f(v.normal)  
            glVertex3f(v.position)  
    glEnd(GL_TRIANGLES)
```

# История графических API: OpenGL 1.0

```
for o in scene.objects:  
    glBegin(GL_TRIANGLES)  
        for t in o.triangles:  
            for v in t.vertices:  
                glColor3f(v.color)  
                glNormal3f(v.normal)  
                glVertex3f(v.position)  
    glEnd(GL_TRIANGLES)
```

- ▶ Данные хранятся в памяти CPU

# История графических API: OpenGL 1.0

```
for o in scene.objects:  
    glBegin(GL_TRIANGLES)  
        for t in o.triangles:  
            for v in t.vertices:  
                glColor3f(v.color)  
                glNormal3f(v.normal)  
                glVertex3f(v.position)  
    glEnd(GL_TRIANGLES)
```

- ▶ Данные хранятся в памяти CPU
- ▶ Несколько OpenGL-вызовов на каждую вершину

# История графических API: OpenGL 1.0

```
for o in scene.objects:  
    glBegin(GL_TRIANGLES)  
    for t in o.triangles:  
        for v in t.vertices:  
            glColor3f(v.color)  
            glNormal3f(v.normal)  
            glVertex3f(v.position)  
    glEnd(GL_TRIANGLES)
```

- ▶ Данные хранятся в памяти CPU
- ▶ Несколько OpenGL-вызовов на каждую вершину
  - ▶ GPU становятся быстрее  $\Rightarrow$  основное время тратится не на рисование, а на накладные расходы самих OpenGL-вызовов

# История графических API: OpenGL 1.0 (1992)

- ▶ Предварительно записать список команд в отдельную сущность - *display list*
  - ▶ Выполнить все команды из списка одной командой `glCallList`
  - ▶ Не любой OpenGL-вызов может быть частью *display list*'а
  - ▶ Предок *command queues* в Vulkan и подобных API

# История графических API: OpenGL 1.0 (1992)

- ▶ Вершины задаются OpenGL-вызовами
- ▶ Display lists
- ▶ Матрицы преобразований вершин: позиционирование объектов и камеры
- ▶ Fixed-function pipeline: настраиваемая, но не расширяемая последовательность операций
- ▶ Асинхронный API: команды выполняются на GPU когда-нибудь

# История графических API: OpenGL 1.1 (1997)

- ▶ Vertex array - спецификация формата и расположения вершин
  - ▶ Нарисовать все вершины одной командой `glDrawArrays`
  - ▶ Вершины всё ещё хранятся на CPU

# История графических API: OpenGL 1.1 (1997)

- ▶ Vertex array - спецификация формата и расположения вершин
  - ▶ Нарисовать все вершины одной командой `glDrawArrays`
  - ▶ Вершины всё ещё хранятся на CPU
- ▶ Текстуры - изображения в памяти GPU, натягиваемые на полигоны

# История графических API: OpenGL 1.2 - 1.4 (1998 - 2002)

- ▶ В текстурах можно записать очень много интересного: normal map, material map, bump map
- ▶ Хочется выполнять сложные вычисления на каждый пиксель
- ▶ ⇒ Texture environments - зачатки программируемости GPU

# История графических API: OpenGL 1.5 (2003)

- ▶ Vertex buffer - возможность хранить вершины в памяти GPU

## История графических API: OpenGL 2.0 (2004)

- ▶ Шейдеры: программы на С-подобном языке GLSL, компилируемые под конкретную GPU
- ▶ Заменяют fixed-function pipeline
  - ▶ Необходимые части fixed-function pipeline остаются

# История графических API: OpenGL 2.0 (2004)

```
// на старте
for o in scene.objects:
    o.createShader()
    o.uploadVertices()

// при рендеринге
for o in scene.objects:
    glUseProgram(o.shader)
    glBindVertexArray(o.vertexArray)
    glDrawArrays(o.vertexCount)
```

# История графических API: OpenGL 3.0 (2008)

- ▶ Огромная часть API объявлена deprecated

# История графических API: OpenGL 3.0 (2008)

- ▶ Огромная часть API объявлена deprecated
- ▶ Transform feedback - возможность записать результат работы шейдеров обратно в вершинный буфер
  - ▶ Зачатки GPGPU

# История графических API: OpenGL 3.1 (2009)

- ▶ Объявленные *deprecated* возможности удалены

# История графических API: OpenGL 3.1 (2009)

- ▶ Объявленные `deprecated` возможности удалены
- ▶ `Instanced rendering` - нарисовать много копий одного объекта в разных местах одной командой

# История графических API: OpenGL 3.2 (2009)

- ▶ Механизм профилей
  - ▶ Core profile
    - ▶ Обязан поддерживаться
    - ▶ Только функционал конкретной версии OpenGL
  - ▶ Compatibility profile
    - ▶ Не обязан поддерживаться
    - ▶ Функционал этой и всех предыдущих версий OpenGL

# История графических API: OpenGL 3.2 (2009)

- ▶ Механизм профилей
  - ▶ Core profile
    - ▶ Обязан поддерживаться
    - ▶ Только функционал конкретной версии OpenGL
  - ▶ Compatibility profile
    - ▶ Не обязан поддерживаться
    - ▶ Функционал этой и всех предыдущих версий OpenGL
- ▶ Геометрические шейдеры - возможность менять тип геометрии и количество вершин на лету

# История графических API: OpenGL 3.3 (2010)

- ▶ Улучшенный instanced rendering
  - ▶ Часть вершинных атрибутов может быть общей для всех копий, другая часть меняться от копии к копии

# История графических API: OpenGL 3.3 (2010)

- ▶ Улучшенный instanced rendering
  - ▶ Часть вершинных атрибутов может быть общей для всех копий, другая часть меняться от копии к копии
- ▶ Texture swizzle

# История графических API: OpenGL 3.3 (2010)

- ▶ Улучшенный instanced rendering
  - ▶ Часть вершинных атрибутов может быть общей для всех копий, другая часть меняться от копии к копии
- ▶ Texture swizzle
- ▶ Нумерация версий языка шейдеров GLSL синхронизирована с версиями OpenGL

# История графических API: OpenGL 4.0 (2010)

- ▶ Шейдеры тесселяции - увеличивают детализацию геометрии на лету
  - ▶ Гораздо меньше возможностей, чем у геометрических шейдеров, зато быстрее

# История графических API: OpenGL 4.0 (2010)

- ▶ Шейдеры тесселяции - увеличивают детализацию геометрии на лету
  - ▶ Гораздо меньше возможностей, чем у геометрических шейдеров, зато быстрее
- ▶ Indirect drawing - можно вычислять количество вершин и их расположение в памяти на лету на GPU, и использовать вычисленные значения для команд рисования

# История графических API: OpenGL 4.1 - 4.7 (2010 - 2017)

- ▶ Compute shaders - настоящее GPGPU внутри OpenGL
- ▶ Проработка и детализация API
- ▶ Атомарные операции в шейдерах
- ▶ Вливание расширений в стандарт OpenGL
- ▶ [www.khronos.org/opengl/wiki/History\\_of\\_OpenGL](http://www.khronos.org/opengl/wiki/History_of_OpenGL)

# История графических API: Vulkan 1.1 (2018)

- ▶ 700 строк кода, чтобы нарисовать один треугольник

# История графических API: Vulkan 1.1 (2018)

- ▶ 700 строк кода, чтобы нарисовать один треугольник
- ▶ Крайне низкоуровневый API
- ▶ Последовательности команд (command queues) в явном виде (display lists done right)
  - ▶ Можно распараллелить генерацию command queues на несколько CPU

# История графических API: Vulkan 1.1 (2018)

- ▶ 700 строк кода, чтобы нарисовать один треугольник
- ▶ Крайне низкоуровневый API
- ▶ Последовательности команд (command queues) в явном виде (display lists done right)
  - ▶ Можно распараллелить генерацию command queues на несколько CPU
- ▶ Похож на DirectX 12, Metal
- ▶ [vulkan-tutorial.com](http://vulkan-tutorial.com)

# Разновидности OpenGL

- ▶ OpenGL

# Разновидности OpenGL

- ▶ OpenGL
- ▶ OpenGL ES (Embedded Systems)
  - ▶ OpenGL ES 1.0 ≈ OpenGL 1.3
  - ▶ OpenGL ES 2.0 ≈ OpenGL 2.0
  - ▶ OpenGL ES 3.0 ≈ OpenGL 3.0

# Разновидности OpenGL

- ▶ OpenGL
- ▶ OpenGL ES (Embedded Systems)
  - ▶ OpenGL ES 1.0 ≈ OpenGL 1.3
  - ▶ OpenGL ES 2.0 ≈ OpenGL 2.0
  - ▶ OpenGL ES 3.0 ≈ OpenGL 3.0
- ▶ WebGL
  - ▶ WebGL 1.0 ≈ OpenGL ES 2.0
  - ▶ WebGL 2.0 ≈ OpenGL ES 3.0

# Разновидности OpenGL

- ▶ OpenGL
- ▶ OpenGL ES (Embedded Systems)
  - ▶ OpenGL ES 1.0 ≈ OpenGL 1.3
  - ▶ OpenGL ES 2.0 ≈ OpenGL 2.0
  - ▶ OpenGL ES 3.0 ≈ OpenGL 3.0
- ▶ WebGL
  - ▶ WebGL 1.0 ≈ OpenGL ES 2.0
  - ▶ WebGL 2.0 ≈ OpenGL ES 3.0
- ▶ OpenGL SC (Safety Critical)
  - ▶ Убраны любые способы отстрелить себе ногу, но в ущерб производительности

# Что такое OpenGL?

- ▶ Не библиотека
- ▶ Спецификация API (документ) на языке C
  - ▶ Описание констант-перечислений (тэгов)
  - ▶ Описание сигнатур функций и их семантики

# Что такое реализация OpenGL?

- ▶ Заголовочный файл, поставляемый системой или драйвером
  - ▶ Определение типов, e.g. `typedef unsigned int GLenum;`
  - ▶ Определение констант, e.g.  
`#define GL_TEXTURE_2D 0x0DE1`
  - ▶ Объявление функций, e.g.  
`void glBindTexture(GLenum target, GLuint texture);`

# Что такое реализация OpenGL?

- ▶ Заголовочный файл, поставляемый системой или драйвером
  - ▶ Определение типов, e.g. `typedef unsigned int GLenum;`
  - ▶ Определение констант, e.g.  
`#define GL_TEXTURE_2D 0x0DE1`
  - ▶ Объявление функций, e.g.  
`void glBindTexture(GLenum target, GLuint texture);`
- ▶ Бинарная реализация объявленных функций (обычно - динамическая библиотека), поставляемая системой и/или драйвером
  - ▶ Может содержать непосредственную реализацию OpenGL как часть драйвера и общаться с GPU
  - ▶ Может быть промежуточным звеном, маршрутизирующим вызов до драйвера
  - ▶ Может быть заглушкой

# Что такое реализация OpenGL?

- ▶ Заголовочный файл
  - ▶ Linux: GL/gl.h - до OpenGL 1.3

# Что такое реализация OpenGL?

- ▶ Заголовочный файл
  - ▶ Linux: GL/gl.h - до OpenGL 1.3
  - ▶ Windows: GL/gl.h - до OpenGL 1.1

# Что такое реализация OpenGL?

- ▶ Заголовочный файл
  - ▶ Linux: GL/gl.h - до OpenGL 1.3
  - ▶ Windows: GL/gl.h - до OpenGL 1.1
  - ▶ MacOS: OpenGL/gl.h - до OpenGL 2.1
    - ▶ Не OpenGL/OpenGL.h
  - ▶ Все платформы: GL/glext.h вместе с  
`#define GL_GLEXT_PROTOTYPES` - до OpenGL 4.6

# Что такое реализация OpenGL?

- ▶ Динамическая библиотека
  - ▶ Linux: libGL.so
  - ▶ Windows: opengl32.dll
  - ▶ MacOS: OpenGL framework

# Что такое реализация OpenGL?

- ▶ Динамическая библиотека
  - ▶ Linux: libGL.so
  - ▶ Windows: opengl32.dll
  - ▶ MacOS: OpenGL framework
- ▶ Может не содержать функции всех версий OpenGL
  - ▶ Под Linux обычно содержит
- ▶ Остальные функции OpenGL нужно динамически загружать специфичными для платформы средствами
- ▶ ⇒ Библиотеки-загрузчики OpenGL

# Загрузчики OpenGL

- ▶ С и C++ specific, для других языков обычно встроено в обёртку над OpenGL
- ▶ [www.khronos.org/opengl/wiki/OpenGL\\_Library](http://www.khronos.org/opengl/wiki/OpenGL_Library)
- ▶ Обычно содержат код, автоматически сгенерированный по XML-спецификации OpenGL
- ▶ Мы будем использовать [GLEW](#)
  - ▶ Но вы можете использовать что угодно!

# Контекст OpenGL

- ▶ Привязан к конкретной реализации OpenGL
- ▶ Привязан к конкретной версии OpenGL
- ▶ Привязан к экрану / окну оконной системы / изображению в памяти
- ▶ Хранит текущее глобальное состояние OpenGL
- ▶ [www.khronos.org/opengl/wiki/OpenGL\\_Context](http://www.khronos.org/opengl/wiki/OpenGL_Context)
- ▶ Создаётся специфичными для платформы средствами
- ▶ ⇒ Библиотеки, создающие контекст OpenGL

# Библиотеки, создающие контекст OpenGL

- ▶ Обычно привязывают контекст к окну и умеют обрабатывать события оконной системы
- ▶ GLUT - устаревшая, плохой интерфейс
- ▶ GLFW
- ▶ SDL2 - умеет загружать изображения, выводить звук, и другое
- ▶ open.gl/context
  - ▶ Можете использовать любую

# Как начать работать с OpenGL?

```
window = createWindow(title)
context = createGLContext(window, version, profile)
context.makeCurrent()
loadGLFunctions()
// тут можно работать с OpenGL!
```

# Литература, ссылки

- ▶ Realtime графика
  - ▶ Computer Graphics: Principles and Practice - книжка начального уровня
  - ▶ Real-Time Rendering (4th edition) - обзор передовых алгоритмов индустрии
  - ▶ GPU Gems 1, 2, 3 - журнал про техники и алгоритмы
- ▶ OpenGL
  - ▶ [www.khronos.org/opengl/wiki](http://www.khronos.org/opengl/wiki) - подробное изложение всех аспектов OpenGL
  - ▶ [docs.gl](http://docs.gl) - удобная документация по отдельным функциям
  - ▶ [learnopengl.com](http://learnopengl.com) - уроки по отдельным темам