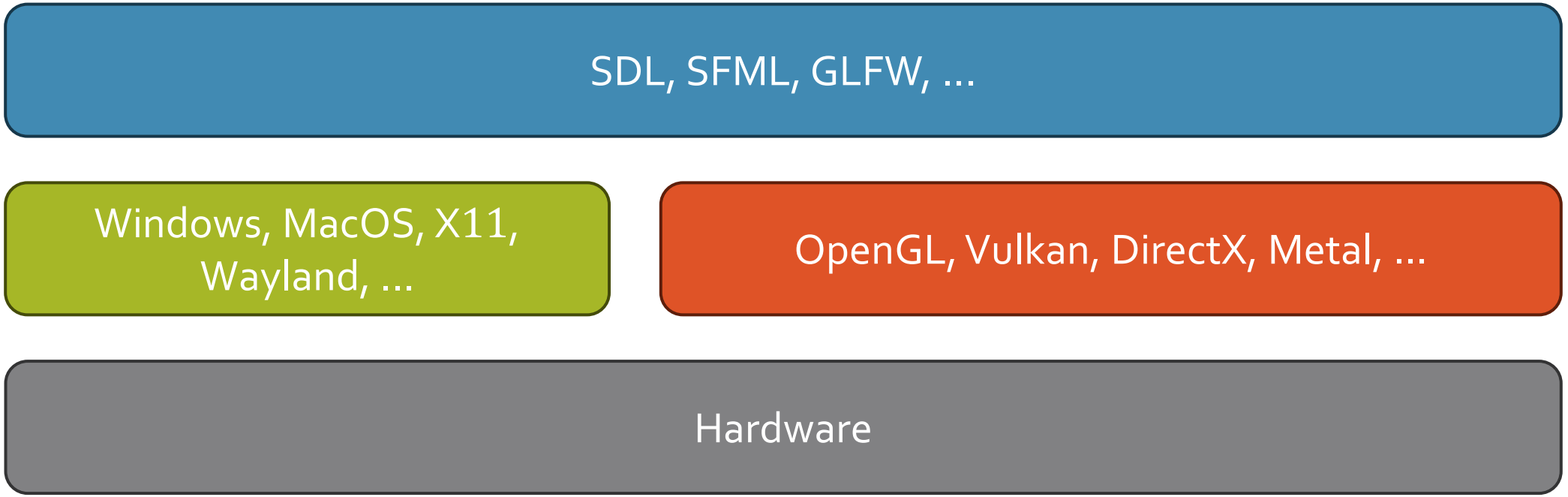
ВИЗУАЛИЗАЦИЯ

Простая визуализация через библиотеку SDL.

К. Владимиров, Yadro, 2024
mail-to: konstantin.vladimirov@gmail.com

Обзор SDL

- Библиотека SDL является одной из распространённых библиотек для простой визуализации.



https://wiki.libsdl.org/SDL2/CategoryAPI

Начинаем работу

- Первое, что нужно сделать это инициализировать библиотеку.

```
res = SDL_Init(SDL_INIT_VIDEO)
```

- Теперь res содержит 0 если всё хорошо и код ошибки если нет.
- В принципе код ошибки можно проверить напрямую.

```
if (res != 0) {
  fprintf(stderr, "sdl init error: %s\n", SDL_GetError());
  abort();
}
```

- Вопрос хотим ли мы это копипастить фактически для каждой функции?

Обработка ошибок

- На помощь может придти макрос вроде такого (обратите внимание на стрингификацию).

```
#define ERR(S) do { \
  const char *Func = #S; \
  fprintf(stderr, "%s error: %s\n", Func, SDL_GetError()); \
  abort(); \
} while (0)

if (SDL_Init(SDL_INIT_VIDEO) != 0)
  ERR(SDL_Init);
```

- В нём, к сожалению, нет возможности управлять важностью проблемы, но его можно доработать.

https://godbolt.org/z/Gd49bavE9

Регистрируем SDL_Quit

- Вызов SDL_Quit в конце очень важен, но хотим ли мы постоянно следить не забыли ли мы её вызвать перед выходом?
- Чтобы не забыть что-то сделать перед выходом из программы в языке C есть механизм atexit.

```
int atexit(void (*func)(void));
```

- Он регистрирует указатель на функцию. Возвращает ноль если регистрация удалась.
- Можно зарегистрировать до 32 функций.

Окно и рендерер

- Окно создаётся через функцию

```
screen = SDL_CreateWindow( /* аргументы */ );
```

- Далее к этому окну можно привязать рендерер.

```
ren = SDL_CreateRenderer(screen, /* аргументы */ );
```

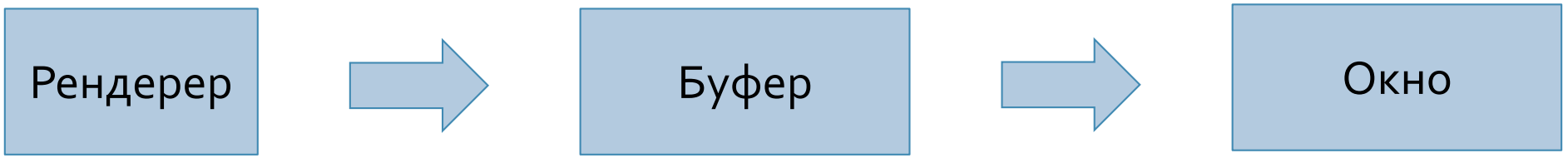
- Ренедер рисует в окно. Сам по себе он привязан к одному из девайсов.
- Большая часть дальнейших вещей делается рендерером.

Получить информацию о девайсах

```
int ndrivers = SDL_GetNumRenderDrivers();
if (ndrivers < 0)
  ERR(SDL_GetNumRenderDrivers);

for (int i = 0; i < ndrivers; ++i) {
  int res; SDL_RendererInfo nfo;

  res = SDL_GetRenderDriverInfo(i, &nfo);
  if (res < 0)
    ERR(SDL_GetRenderDriverInfo);
  else
    printinfo(i, &nfo);
}
```

Двойная буферизация

- В библиотеке SDL двойная буферизация идёт по умолчанию.



SDL_RenderClear,
SDL_RenderGeometry,
… etc …

SDL_RenderPresent
Отображает старый буфер на окно.
Заменяет буфер на новый для рендеринга.

Цикл поллинга

- Базовый вариант.

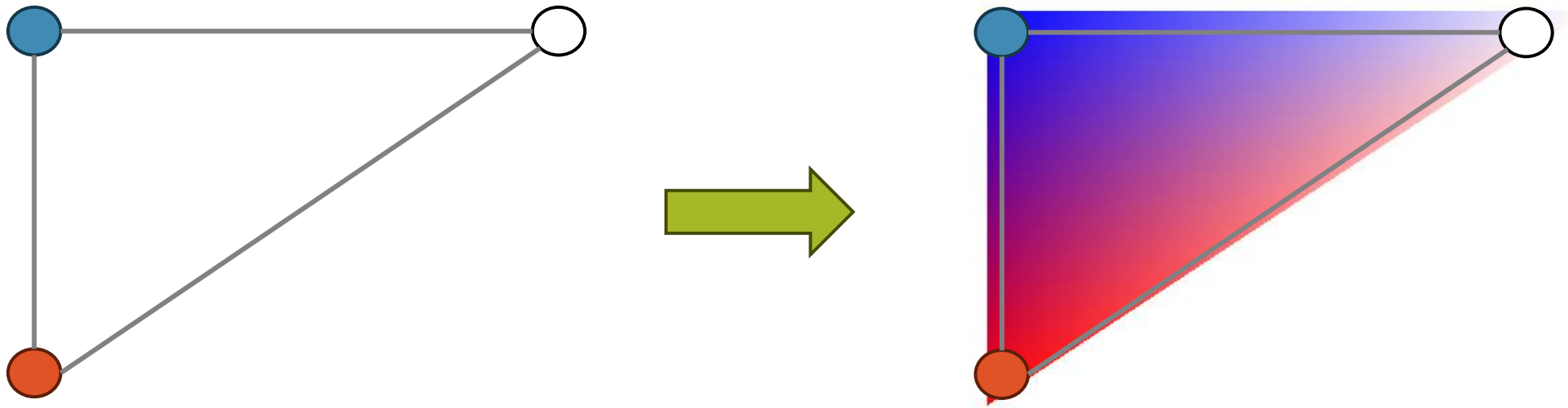
```
for (;;) {
  int pending;
  SDL_Event event;
  pending = SDL_PollEvent(&event);
  if (pending && event.type == SDL_QUIT)
    break;
  renderTriangle(v.ren);
  SDL_RenderPresent(v.ren);
}
```

- Конечно лучше разобраться со всеми накопившимися событиями прежде чем переходить к следующему кадру.

Простая геометрия и рендринг

- Простая геометрия интерполируется и растеризуется.

```
SDL_Vertex vertices[] = {vertex_1, vertex_2, vertex_3};
SDL_RenderGeometry(ren, NULL, vertices, 3, NULL, 0);
```



sdl_triangle.c

Работа с текстурами

- Текстура может быть довольно легко загружена из файла.

```
tex = IMG_LoadTexture(ren, "dragon.png");
```

- Заметим, мы должны использовать рендерер чтобы создать текстуру.
- Далее рендерер просто копирует кусок текстуры в нужный квадрат.

```
SDL_Rect dst = { 100, 100, 256, 256 };
SDL_RenderCopy(ren, tex, NULL, &dst);
```

- Интересно, что текстуры можно формировать программно, нацеливая на них рендерер и рендеря туда как в окно.

sdl_texture.c

Контроль частоты кадров

- Возможно нам не хочется перерисовывать экран слишком часто.

```
for (;;) {
  SDL_Event event;
  Uint32 start, elapsed, estimated = 1000 / 50;
  start = SDL_GetTicks();

  // тут какой-то рендеринг и обработка событий

  SDL_RenderPresent(v.ren);
  elapsed = SDL_GetTicks() - start;
  if (elapsed < estimated)
    SDL_Delay(estimated - elapsed);
}
```

Пересчёт координат в экранные

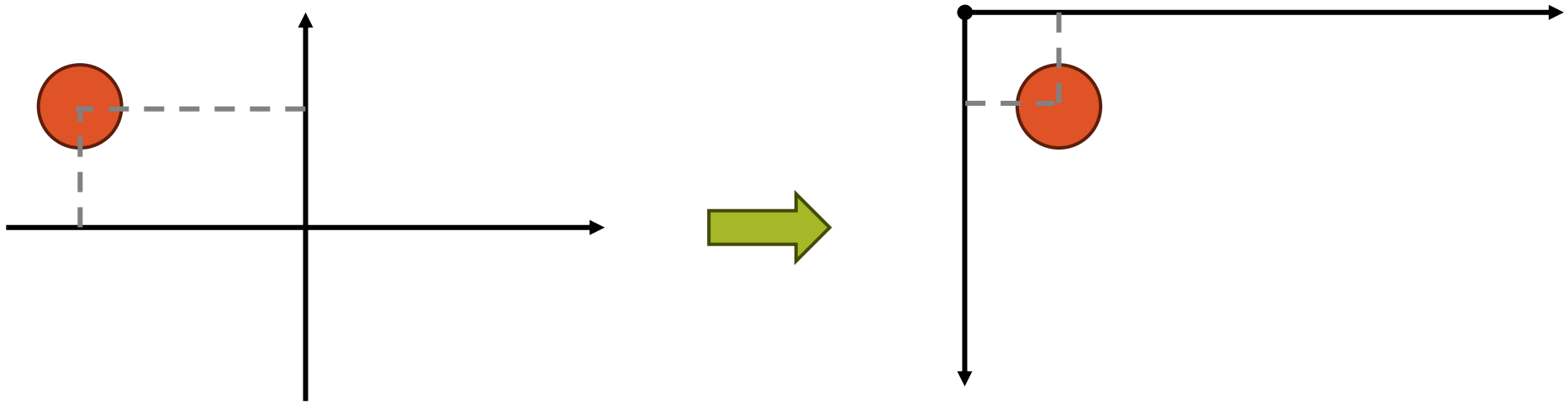
- Для правильной оценки скоростей проще всего находиться в логической системе координат.

```
SDL_Point to_physical(SDL_Renderer *r, double x, double y) {
  int width, height; SDL_Point p;
  SDL_GetRendererOutputSize(r, &width, &height);
  double w = width, h = height;
  p.x = ((x + 1.0) * w / 2.0); p.y = ((-y + 1.0) * h / 2.0);
  return p;
}
```

- Посмотрите на эту функцию и давайте осознаем как она переводит логические координаты в экранные.

Пересчёт координат в экранные

- Для правильной оценки скоростей проще всего находиться в логической системе координат и в экранную уже пересчитывать.

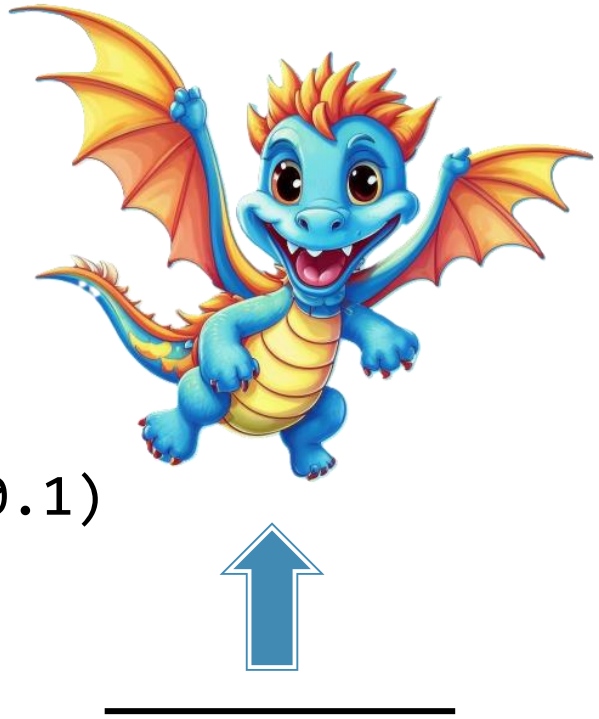


sdl_animation.c

Работа с кнопками

- SDL позволяет делать вещи довольно интерактивными.

```
while (SDL_PollEvent(&event)) {
  if (event.type == SDL_QUIT)
    goto cleanup;
  if (event.type == SDL_KEYDOWN) {
    SDL_Keycode kc = event.key.keysym.sym;
    if (kc == SDLK_UP && ypos <= ground_level + 0.1)
      speed = start_speed;
  }
}
```



- Теперь можно строить логику обработки нажатий кнопок.

sdl_texture.c

Внезапная сложность с текстом

- Текст на экран нужно тоже отрендерить, то есть как минимум:
- Загрузить шрифт.

```
TTF_Font* font = TTF_OpenFont("arial.ttf", size);
```

- Отрендерить в текстуру.

```
surface = TTF_RenderText_Solid(font, "Hello, world!", *color);
texture = SDL_CreateTextureFromSurface(renderer, surface);
```

- И далее эту текстуру уже отобразить на экран.
- Для статического текста это ок, но для динамического текста это так себе.

Обсуждение: bitmap fonts

- Допустим у вас есть картинка с цифрами, буквами и прозрачным фоном.
- Как бы вы организовали хранение и работу с таким заранее заготовленным "шрифтом"?

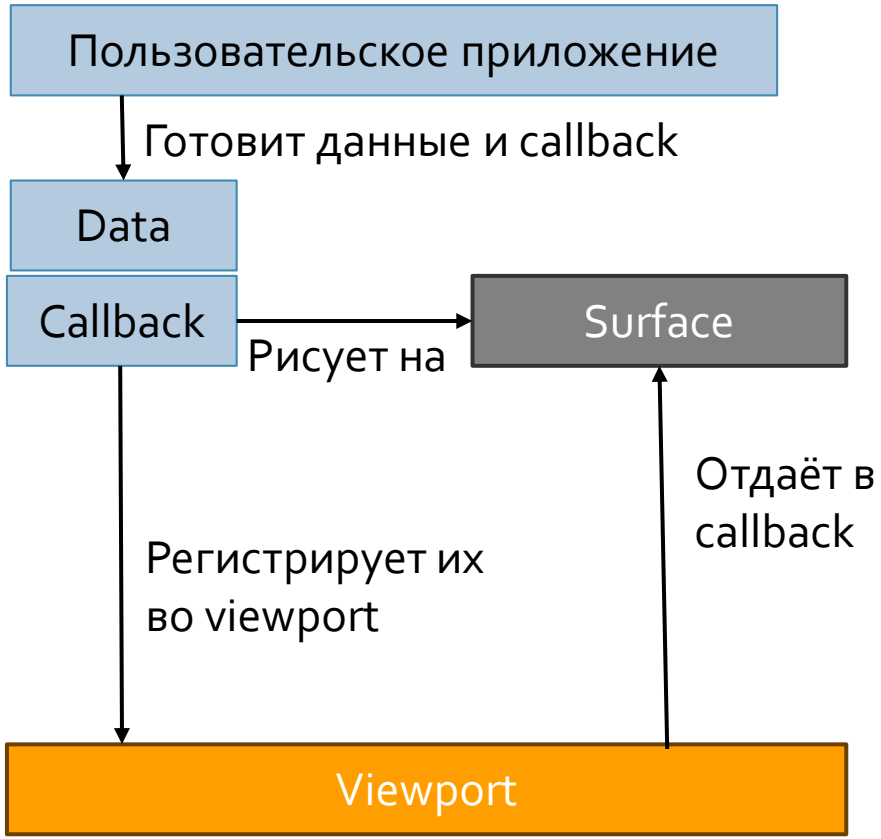
К абстрагированию рисовалки

- Предположим, что мы не хотим закладываться даже на детали SDL.

```
struct Surface;

void Surface_fillwith( /* аргументы */ );
void Surface_putpixel( /* аргументы */ );

struct ViewPort;

typedef void (*callback_t)(struct Surface *, void *);

enum pollres ViewPort_poll(struct ViewPort *v);
struct ViewPort *ViewPort_query(/* аргументы */,
  callback_t callback, void *data);
```

Верхний уровень архитектуры

- Viewport
  - Абстрагирует polling.
  - Содержит и зовёт callbacks.
  - Формирует surface для callbacks.
- Surface
  - Абстрагирует рендеринг.
  - Предоставляет контекст для рисования.



drawer.h

Пример: рисовалка для Julia set

```
struct julia_data {
  complex double *pc;
  complex double *pcenter;
  double *psz;
};

static void draw_julia(struct Surface *s, void *data) {
  struct julia_data *jd = (struct julia_data *)data;
  // тут всё делается в терминах s и jd
}
```

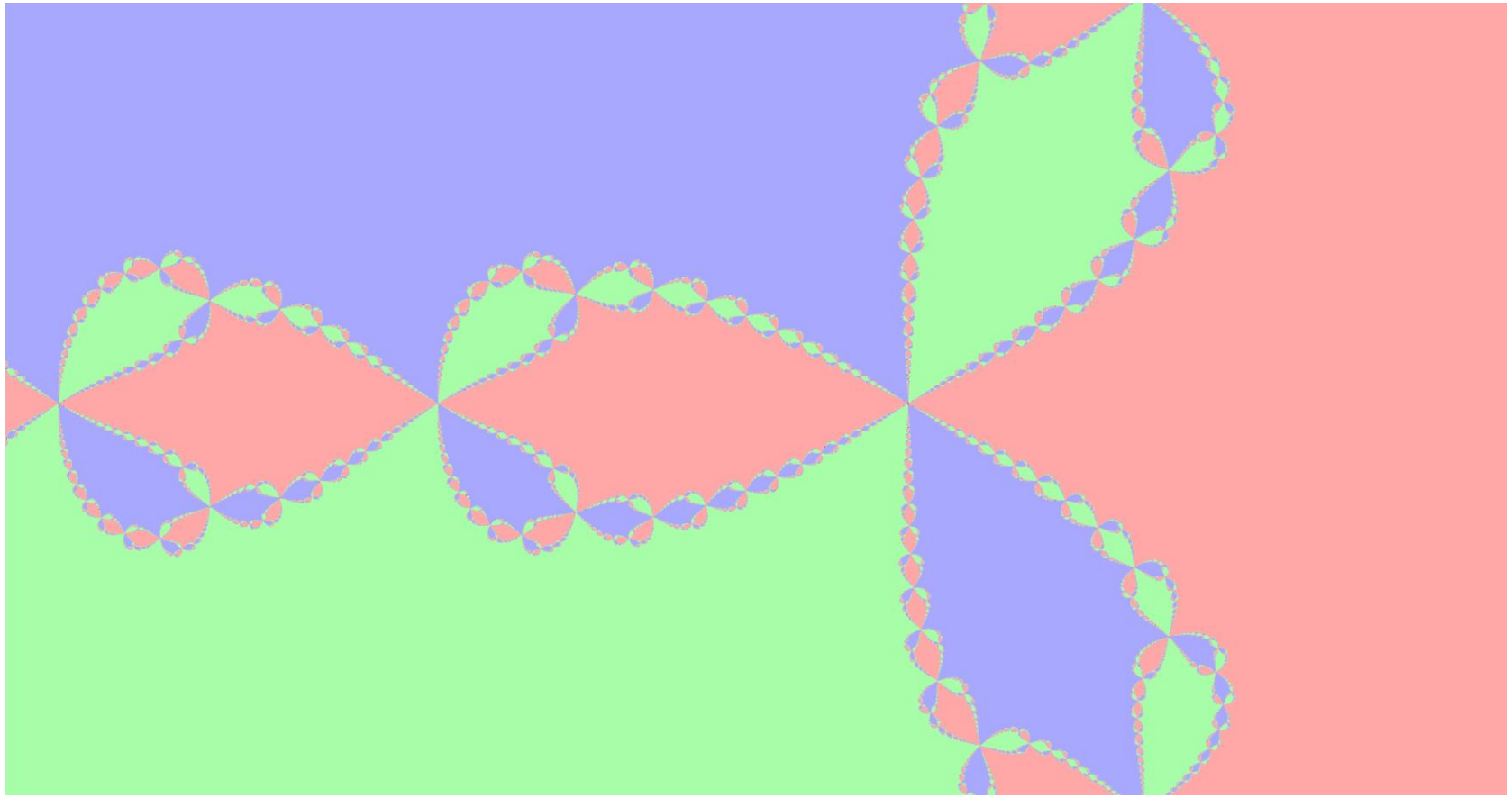
Пример: рисовалка для Julia set

```
struct julia_data { /* данные */ };

static void draw_julia(struct Surface *s, void *data) {
  struct julia_data *jd = (struct julia_data *)data;
  // тут всё делается в терминах s и jd
}

struct julia_data jd = {&c, &center, &sz};
struct ViewPort *v = ViewPort_query(x, y, draw_julia, &jd);

while (ViewPort_poll(v) == PROCEED) {
  arg += argmul * dphase;
  c = CMPLX(abs * cos(arg), abs * sin(arg));
}
```

Добавим паузу и скриншоты

```
SDL_SetRenderTarget(ren, texture); // переключим на текстуру

callback(ren, data); // отрисовка в текстуру

SDL_QueryTexture(v->texture, NULL, NULL, &width, &height);

SDL_Surface *surface =
      SDL_CreateRGBSurface(0, width, height, 32, 0, 0, 0, 0);
SDL_RenderReadPixels(ren, NULL, surface->format->format,
                     surface->pixels,surface->pitch);
SDL_SaveBMP(surface, name); // сохраним скриншот
SDL_FreeSurface(surface);

SDL_SetRenderTarget(ren, NULL); // вернём на экран
```

Можно ползать по фракталу Ньютона



Задание (самостоятельно)

- Попробуйте написать визуализатор чего-то из тех алгоритмов, которые мы проходили в этом курсе.
- Например визуализатор LRU кеша: кеш находится посередине, ловит падающие сверху числа, при кеш-хите загорается коробочка, а вытесняемые числа улетают вниз.
- Или что-то ещё в этом духе. Визуализации это редкостное удовольствие.