Simple DirectMedia Layer SDL2

Prof. Raoul Bidouille

http://dept-info.labri.fr/~namyst/no_link/soft/

Go and download NOW!

What do you need to attend this tutorial?

- A laptop with enough battery capacity
- A decent operating system
  - If you're under Linux, make sure you have the latest kernel version (2.4.x)
- libsdl2-devel, libsdl2-image, lidsdl2-mixer
- Mastering trigonometry
- Perseverance, Selflessness, Patience
  - Tutorial is approx. 5 hours long
- A working internet connection

http://dept-info.labri.fr/~namyst/no_link/soft/

What will we learn today?

- Use SDL2 to
  - Display and move graphical objects on screen
    - Using a accelerated rendering engine
  - Give the illusion objects are animated
  - Play sounds and music
  - Read keyboard input
  - Use timers to trigger events at specific time

How the heck will we do that?

- Develop a simple 2D game from scratch
- Incrementally add features
  - From v_1 to v_{10}
  - Each version focuses on a particular topic
  - You can implement v_n event if you failed to implement v_{n-1}
  - Shame on you!

Version 0: Curse of the Black Screen

- Look into src/graphics.c
  - SDL_CreateRenderer()
- graphics_render() is called in a loop, as fast as possible
  - SDL_RenderClear()
  - Then draw your scene
  - SDL_RenderPresent()
- SDL implements double buffering

```
cd v00
make
./game
```

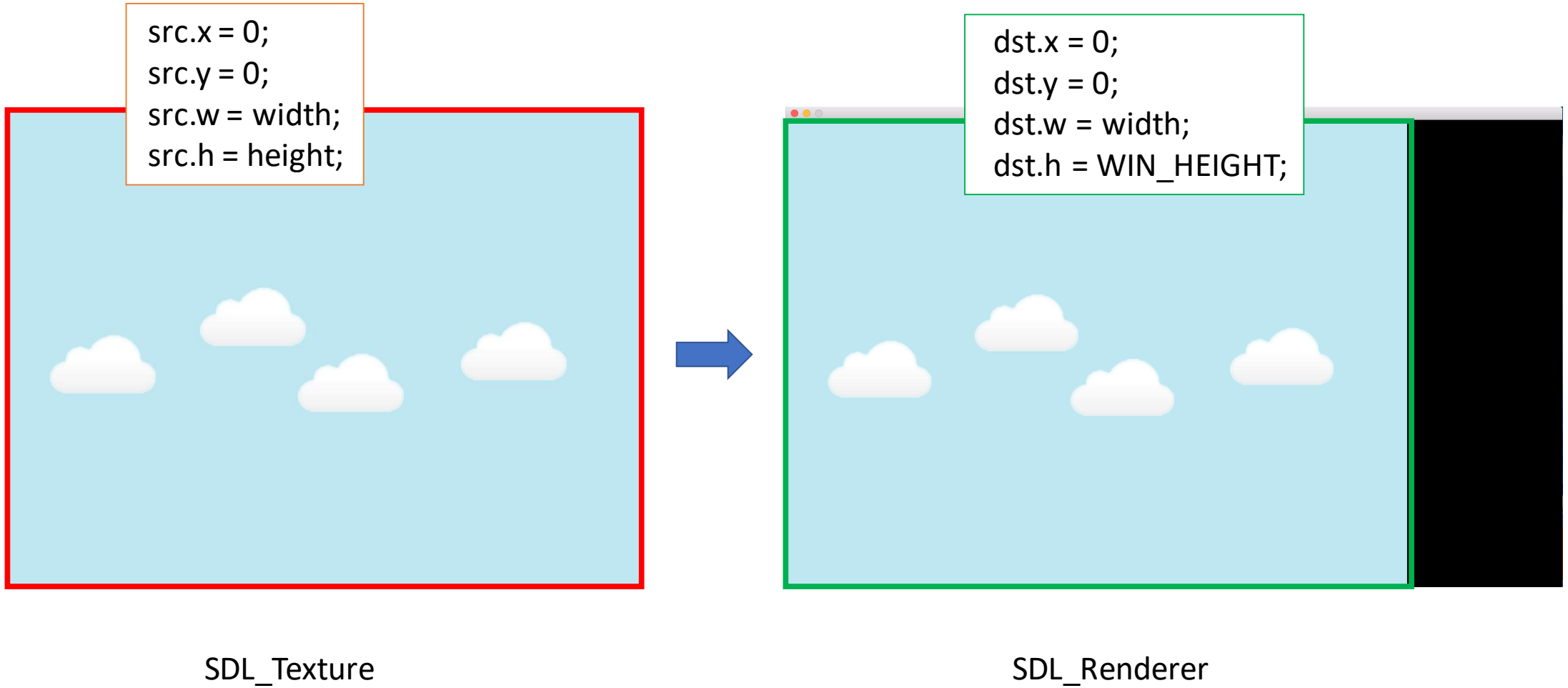
Also try with performance monitoring:

```
./game –d p
```

Version 1: Clouds

- We want clouds on our screen
- We have to
  - Load an image
  - Convert the image to a texture (think “on the gpu”)
  - Stamp our texture on the screen

Version 1: SDL_RenderCopy()



Version 1: Do it!

- Add a loop in graphics_render_background() to fill the screen

- Add a bird, by simply using

  graphics_render_background (bird);

Version 1: Do it!

- Add a loop in graphics_render_background() to fill the screen

- Add a bird, by simply using

  graphics_render_background (bird);

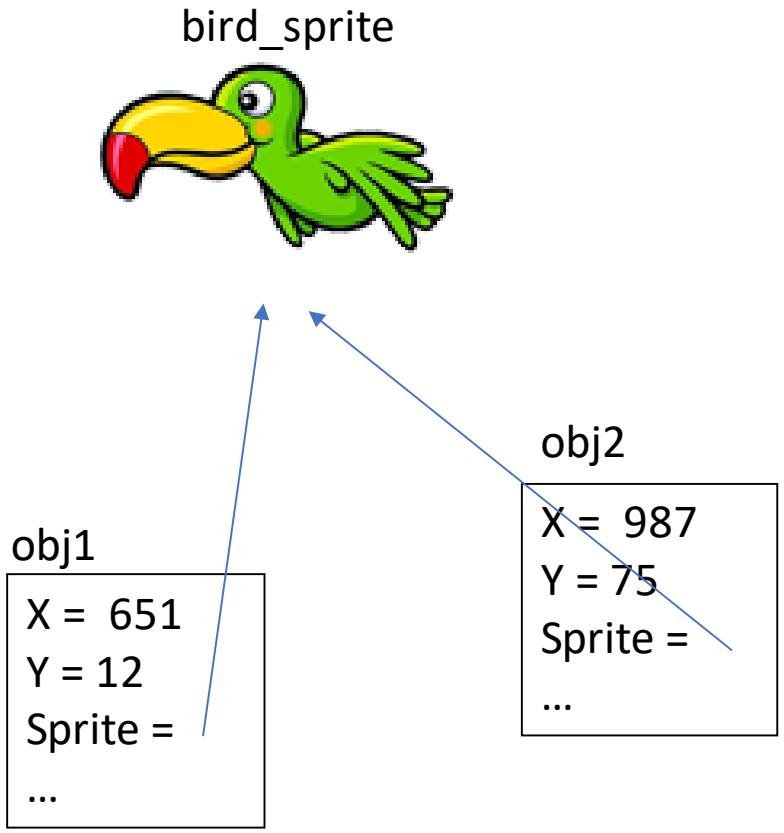
- Looks professional, doesn’t it?

Version 2

- We ultimately want to move the bird, but also many other objects…
- A typical main loop looks like:
  - Poll events (keyboard, mouse, timers) and call appropriate functions
  - Move main character (bird)
  - Animate dynamic objects (missiles, bad birds, explosions, etc.)
  - Compute collisions
  - Render background + objects

Version 2: sprites and objects

- Sprites store graphical representation
  - texture(s), display size
  - It’s a read-only object
  - See src/sprite.c
- Objects
  - Have their own coordinates
  - May share a sprite with other objects



Version 2: Do it!

- Note the unconventional handling of UP, DOWN,LEFT and RIGHT keys
- Go to src/bird.c, and make it move up and down!
- Notice when bird is moved, and when graphical representation is updated..

Version 3: The exhilarating sensation of speed

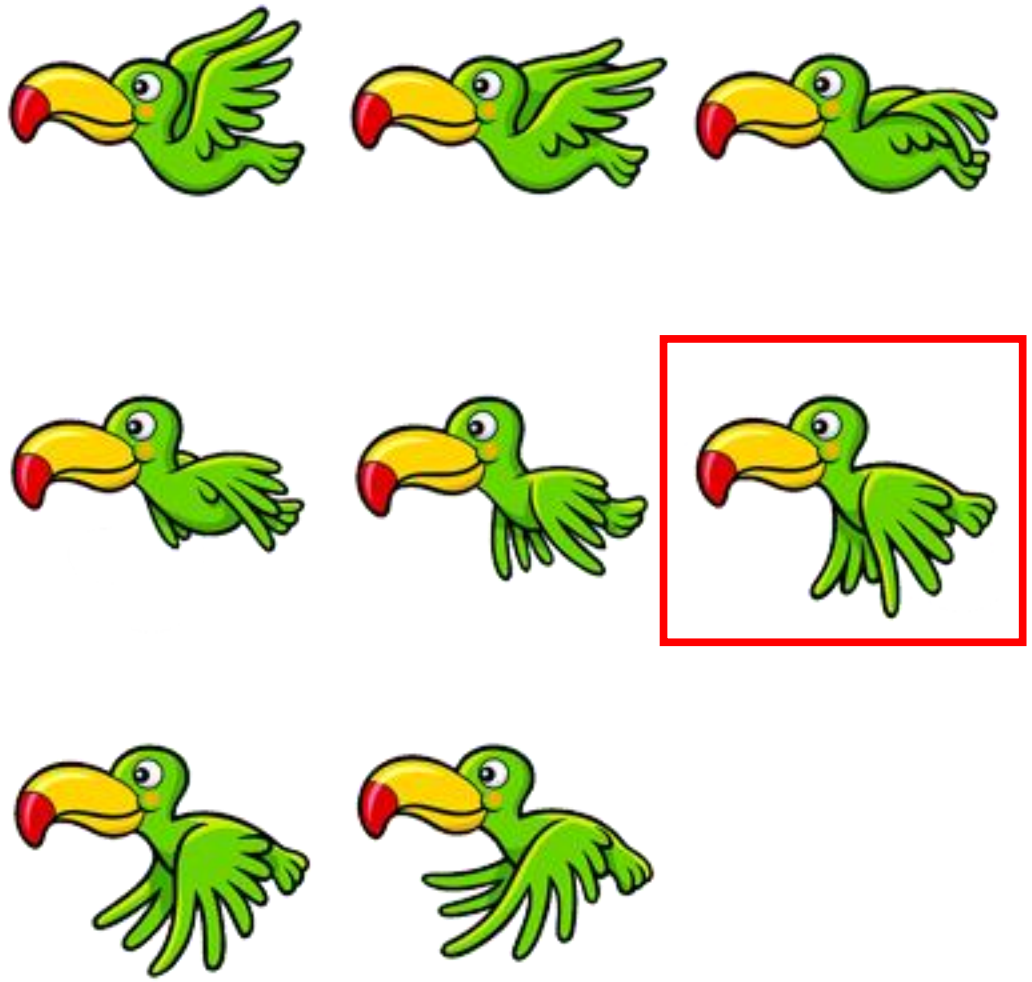
- We will add trees in the background, and make them move along x-axis
  - Bird will stay at a fixed x position

- Trick
  - We will use three different layers
  - Each layer will move a different speed
    - Near layer will move at x-speed = - BIRD_SPEED
    - Middle layer will move at x-speed = - BIRD_SPEED / 2
    - Far layer will move at x-speed = - BIRD_SPEED / 4

- How to make these layers move?
  - Use a changing x-offset when stamping the textures
  - graphics_render_scrolling_background()
  - A story of “least common multiple”…

Version 4: animated sprites

- We now use textures which contain “tilesets”
- The sprite type now contains
  - frames (e.g. 8)
  - xframes (e.g. 3)
  - See src/sprite.c
- Look at graphics_render_object()
  - current_sprite is used to compute the source rectangle



Version4: Do it!

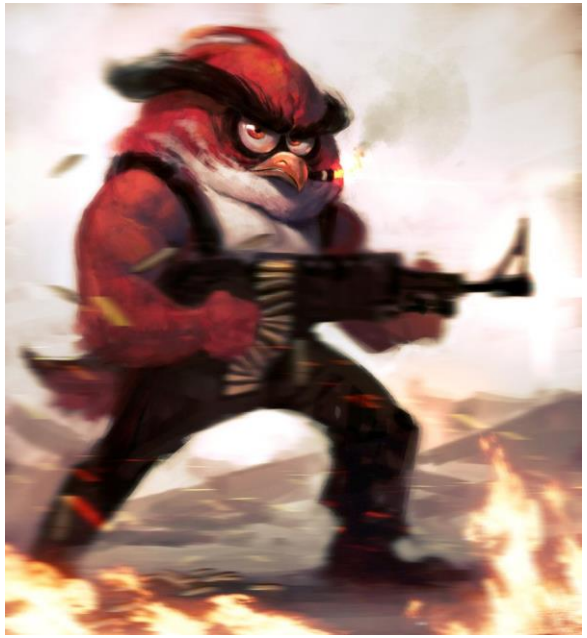
- Go to bird.c and make current_sprite evolve:
  - 0, 1, …, 6, 7, 6, …, 1, 0, 1, …

Version 5: Gravity

- We now want to our bird to become subject to gravity
  - Obj->ys++ should be ok
- Go to src/bird.c and proceed!
- Oh wait, don’t forget to keep the bird within the limits
  - [0, WIN_HEIGHT]

Version 6: It’s wasn’t my war! (First Blood, 1982)

- Our bird goes wild… and wants to shot laser beams
  - Src/missile.c
- Press “SPACE” to shoot
- Hmmm, maybe we need to add some code in animation_missile_onestep()



Version 7: Sound!

- Add play_sound (SOUND_SHOT) in animation_missile_add()
  - Test it!

- Look at src/sound.c
  - Add a call to
    - Mix_PlayMusic (music [current_track], -1);
  - Add a call to
    - animation_notice_add (current_track);

Version 8: Bad birds are coming!

- We use a “generator” object which has no graphical representation on screen
  - Shows how to use timers
- Look at messages in terminal while the game runs…
- Go to src/generator.c
  - Replace printf with animation_bad_add (obj)

Version 8: Bad birds are coming!

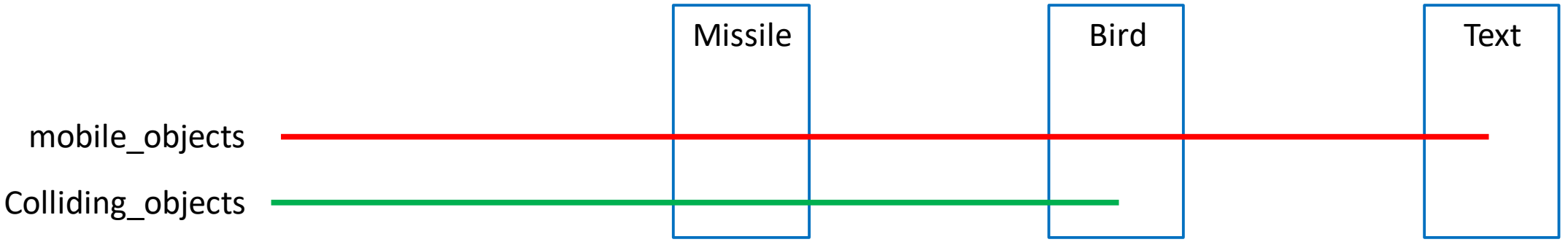
- But for now they follow a stupid trajectory…
  - Go to src/bad.c and use a random altitude

- Oh my god, missiles have no effects ☹

Version 9: Collision detection

- SDL doesn’t help with collision detection
- We will implement a simple, approximated method:
  - Checking bounding box collision
  - animation_check_collision_approx()

- Note: to avoid detect unnecessary collisions (e.g. with notices), collidable objects are stored in an extra list



Version 10: Happiness is easy

- Per pixel collision detection
  - Bitmaps are generated at sprite creation
- Much better uh?
- Thanks for attending!