

Programmieren 1

Graphics, Animations, Reactive Programs
(optional topic)

Graphical Objects: Bitmap Images

- Graphical objects are represented as arrays
- bitmap: [bitmap: "http://.../cat.jpg"]



- Preload image data:


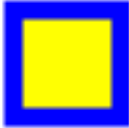

```
"http://...cat.jpg" read-image-url data!  
[bitmap: data] show-image
```

- Show images with show-image
- Example:

```
[bitmap: "https://upload.wikimedia.org/wikipedia/commons/thumb/e/e5/Blue-eyed_domestic_cat_%28Felis_silvestris_catus%29.jpg/192px-Blue-eyed_domestic_cat_%28Felis_silvestris_catus%29.jpg"] show-image
```

Approach inspired by the [Racket Universe Library](https://github.com/leibniz-hannover/racket-universe-library)

Graphical Objects: Squares, Rectangles, Circles, and Ellipses

- square: [square: 40 "red" "black"] show-image 
 [square: 40 "yellow" [pen: "blue" 7]] show-image 
 [square: 40 [color: 0.8 0.8 0.8]
 [pen: [color: 1 0 0] 3]] show-image 

- color: name or red-green-blue components (0.0-1.0 or 0-255)

- rectangle: [rectangle: 40 10 "cyan"] show-image 

- circle: [circle: 40 "red" "black"] show-image 

- ellipse: [ellipse: 40 10 "yellow" "blue"] show-image 

Graphical Objects: Text and Fonts

```
[text: "hello" [font: "Arial" 36] "black"] show-image
```

hello

```
[text: "hello" [font: "Comic Sans MS" 36] "black"] show-image
```

hello

```
[text: "hello" [font: "Comic Sans MS" 36] "lime" "black"] show-image
```

hello

Graphical Objects: Scaling and Rotation

[scale: 2.5 [rectangle: 40 10 "yellow" "blue"]] show-image 

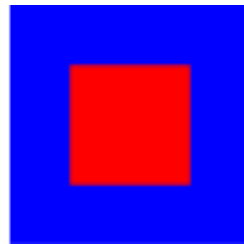
[rotate: -45 PI * 180 / [rectangle: 40 10 "yellow" "blue"]] show-image



Combining Graphical Objects

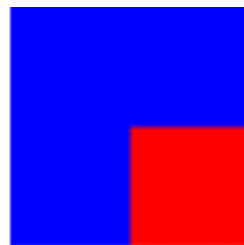
- Combinations of graphical objects are represented as nested arrays

```
[place-image: [square: 50 "red"] [square: 100 "blue"] 50 50]
```



center position of red
square in blue square

```
[place-image: [square: 50 "red"] [square: 100 "blue"] 50 50 "left" "top"]
```




left-top position of red
square in blue square

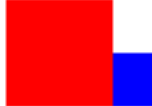
Combining Graphical Objects

- Combinations of graphical objects are represented as nested arrays


```
[beside: [[square: 20 "red"] [square: 20 "blue"]]]
```



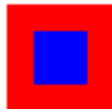
```
[beside: [[square: 40 "red"] [square: 20 "blue"]] "bottom"]
```



```
[above: [[square: 40 "red"] [square: 20 "blue"]] "left"]
```

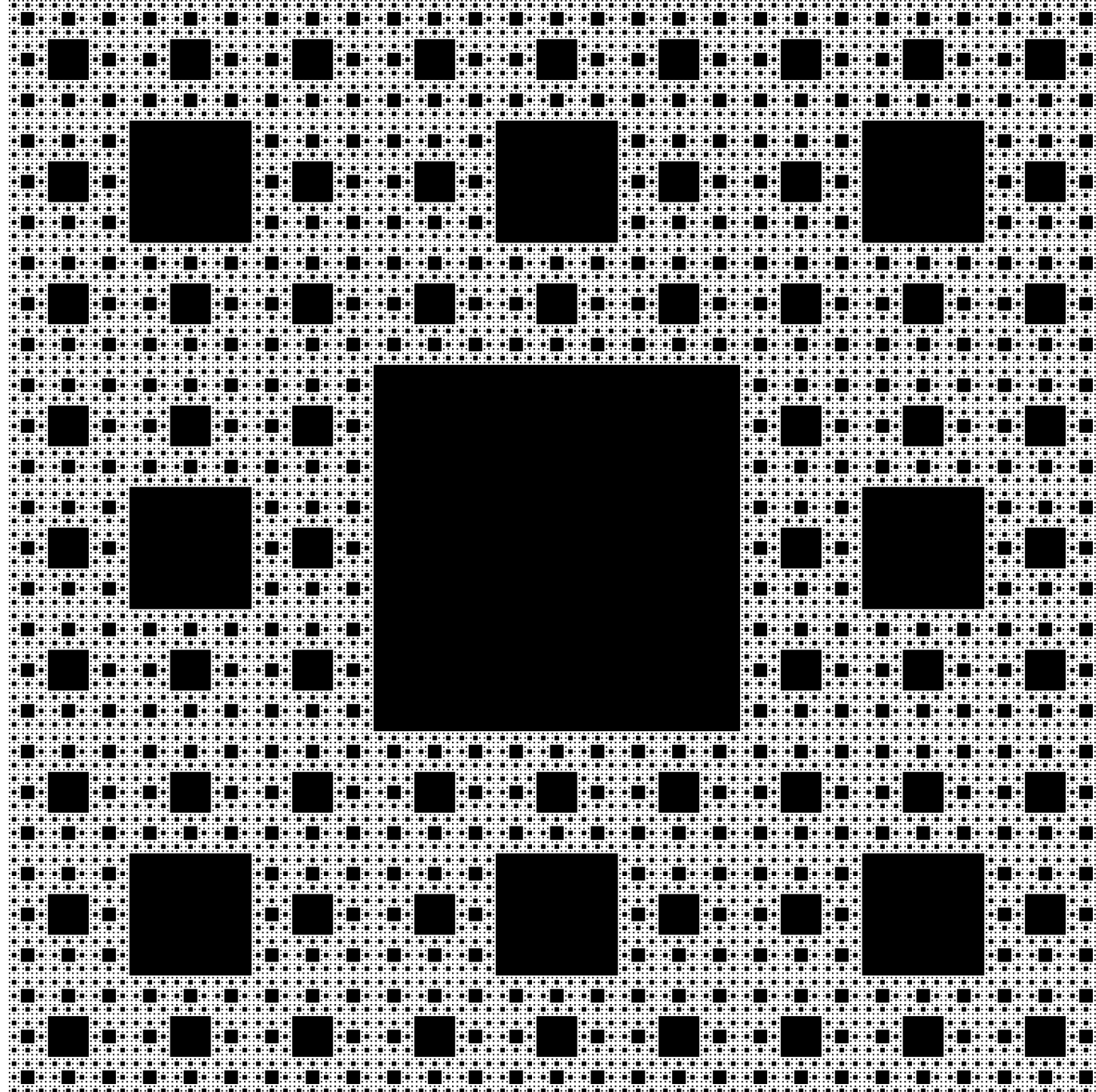


```
[overlay: [[square: 20 "blue"] [square: 40 "red"]]]
```



```
[underlay: ...]
```

Sierpinski Carpet – Recursive Graphics

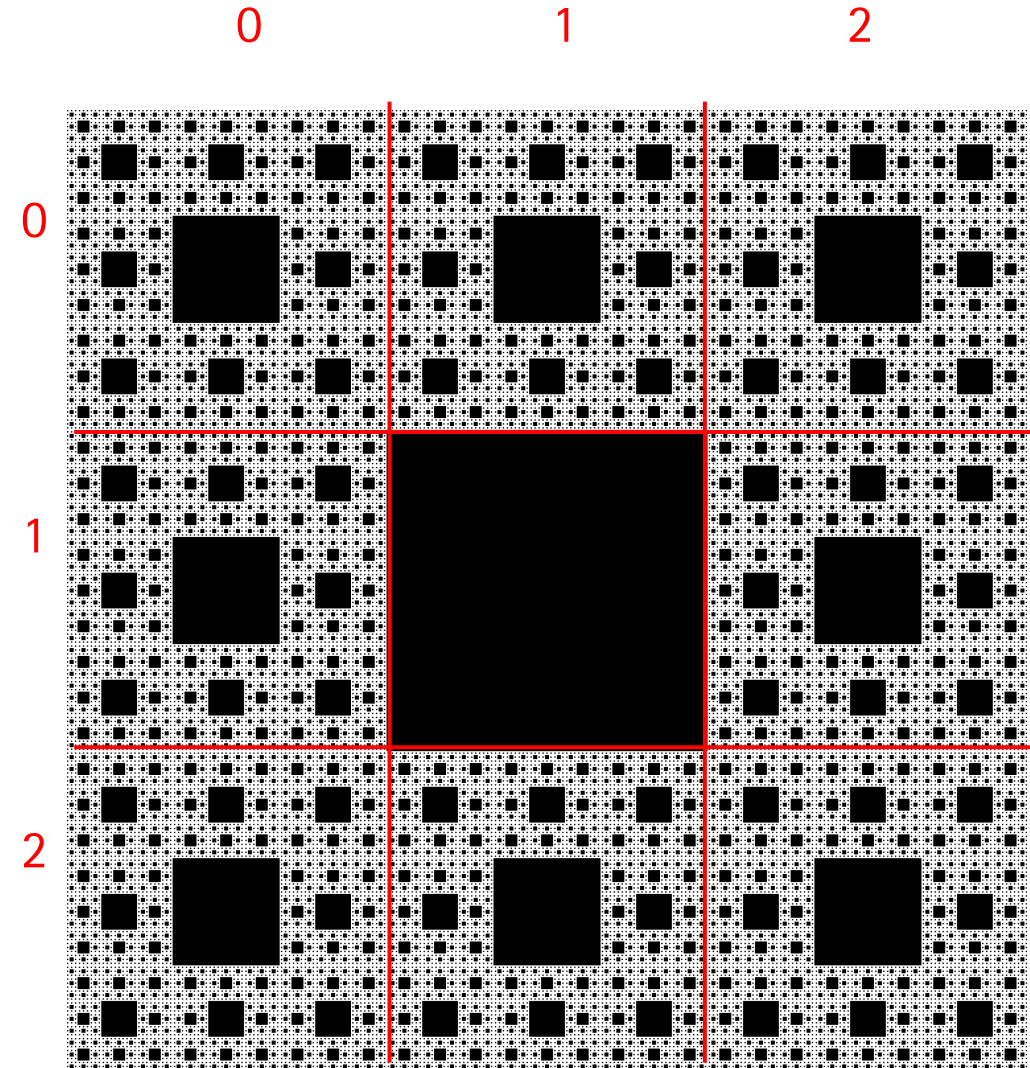


Sierpinski Carpet – Recursive Graphics

```
sierpinski: (d :Int, w :Num -> :Arr) {
  [square: w "white"] bg!
  [square: w 3 / "black"] b!
  d 0 > w 9 >= and {
    d 1 - w 3 / sierpinski s!
    [above: [
      [beside: [s s s]]
      [beside: [s b s]]
      [beside: [s s s]]]]
  } {
    [overlay: [b bg]]
  } if
} fun
```

recursive call

```
5 900 sierpinski show-image
```



Animated Graphics

System operation (pseudo code):

```
state := initial_state
loop {
    # get image for state
    image := on_draw(state) # on_draw is user-defined
    # show image on screen
    draw_internal(image)
    wait(1/60 sec)
    # update state
    state := on_tick(state) # on_tick is user-defined
}
```

Animated Graphics

```
# State -> ImageArray
# Takes the current state and returns an image array.
on-draw: (theta :Num -> :Arr) {
  [rotate: theta [square: 50 "red"]]
} fun

# State -> State
# Current state to next state. Called 60 times per second.
on-tick: (theta :Num -> :Num) { # here, state is a number
  theta 0.03 +
} fun

"Square" 200 200 0 [ # title, width, height, init-state
  on-draw: { on-draw } lam # save draw function
  on-tick: { on-tick } lam # save tick update function
] show
```

Animated Graphics

```
"Title" 200 200 0 [  
  on-draw: (theta) {  
    [rotate: theta [square: 50 "red"]]  
  } lam  
  on-tick: (theta) {  
    theta 0.03 +  
  } lam  
] show
```

Interactive Graphics (Mouse Position)

```
# State -> ImageArray
# Takes the current state and returns an image array.
number->image: (theta :Num -> :Arr) {
  [rotate: theta [square: 50 "red"]]
} fun

# State, String -> State
# Returns next state depending on the mouse position.
on-mouse-move: (state :Num, x :Num, y :Num -> :Num) {
  x 0.03 *
} fun

"Square" 200 200 0 [ # title, width, height, init-state
  on-draw: { number->image } lam
  on-mouse-move: { on-mouse-move } lam
] show
```

Interactive Graphics (Arrow Keys)

```
# State -> ImageArray
number->image: (state :Arr -> :Arr) {
    state .0 theta!
    [rotate: theta [square: 50 "red"]]
} fun
```

```
# State -> State
on-tick: (state :Arr -> :Arr) {
    state .0 theta!
    state .1 delta!
    [theta delta +,    delta]
} fun
```

Interactive Graphics (Arrow Keys)

```
# State, String -> State
on-key-press: (state :Arr, key :Str -> :Arr) {
  { key "ArrowLeft" = } { [state .0,   -0.03] }
  { key "ArrowRight" = } { [state .0,   0.03] }
  { true } { [state .0,   0] }
} cond-fun

# title, width, height, initial-state, array of lambdas
"Square" 200 200 [0 0] [
  on-draw: { number->image } lam
  on-tick: { on-tick } lam
  on-key-press: { on-key-press } lam
] show
```

Callback Functions of "show" Function

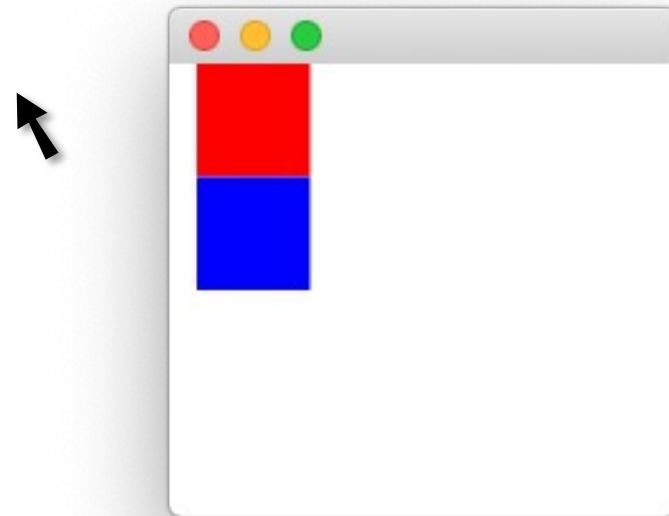
```

"Title" 200 200 0 [
  on-draw: (state) {
    [rotate: state [square: 50 "red"]]
  } lam
  on-tick: (state) { state 0.03 + } lam
  stop-when: (state) { state 6.28 > } lam
  on-key-press: (state key) { 0 } lam
  on-key-release: (state key) { state } lam
  on-mouse-press: (state mouse-x mouse-y) { state } lam
  on-mouse-release: (state mouse-x mouse-y) { state } lam
  on-mouse-move: (state mouse-x mouse-y) { state } lam
  on-mouse-drag: (state mouse-x mouse-y) { state } lam
] show

```


Callback Function on Square Objects

```
"Title" 200 200 0 [
  on-draw: (state) {
    [above: [
      [square: 50 "red" on-mouse-press: { "click red" println } lam]
      [square: 50 "blue" on-mouse-press: { "click blue" println } lam]]
    } lam
  ] show
```



```
click red
click blue
click blue
click red
click red
...
```

Buttons

```
button: (label :Str -> :Arr) {
  [overlay: [[text: label 24 "black"]
    [rectangle: 140 40 "lightgray" "darkgray"]]]
  on-mouse-press: (state :Str x y -> :Str) {
    "clicked " label +
  } lam]
} fun
```



```
on-draw: (state :Str -> :Obj) {
  [above: [[beside: ["OK" button, [square: 20 "white"], "Cancel" button]]
    [text: state 24]] "center"]
} fun
```

```
"Button" 300 100 "Click a button, please" [
  on-draw: { on-draw } lam
] show
```