Lecture #01

2D Graphics

Computer Graphics

Winter term 2016/17

Marc Stamminger

Today

- Basics of 2D-Graphics:
  - Images
  - Framebuffer
  - Graphics APIs
  - Graphics Primitives
  - Scene Graphs

- Lots of examples!

Images

- An image is a 2D-array of pixels (Pixel = **pic**ture **el**ement)
- Each pixel can be
  - a bit: black / white
  - an integer value (typically 8 bits): grey value
  - an integer triple (typically 3x8 bits): color value (red, green and blue)
    → see next lecture “Colors”
  - an integer quadruple (typically 4x8 bits): color value with transparency
  - float instead of integer: high dynamic range images (HDR images)

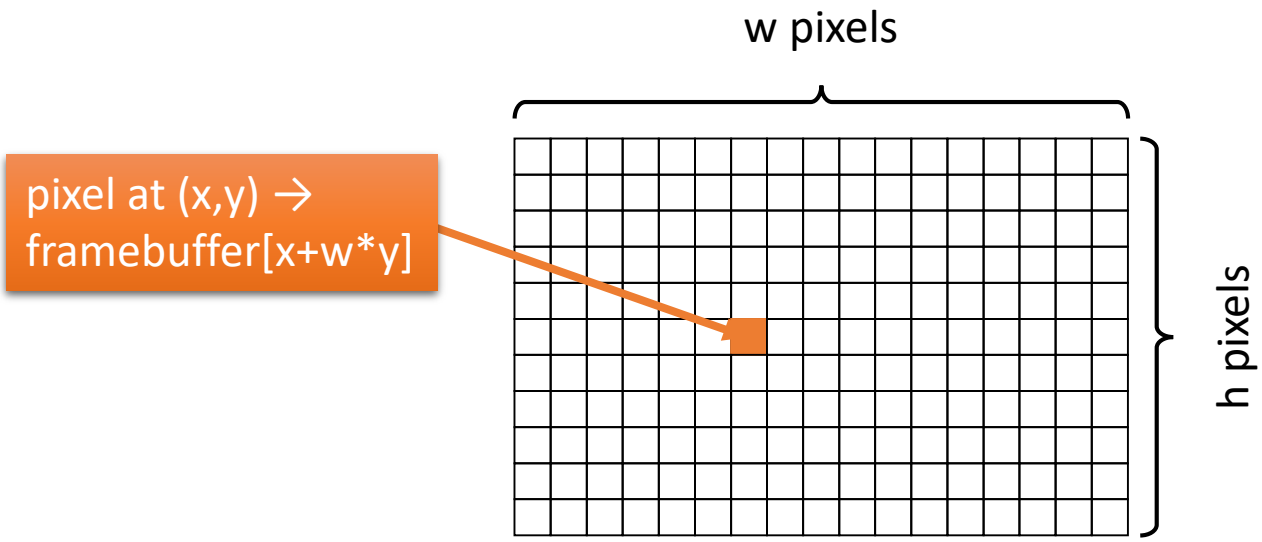


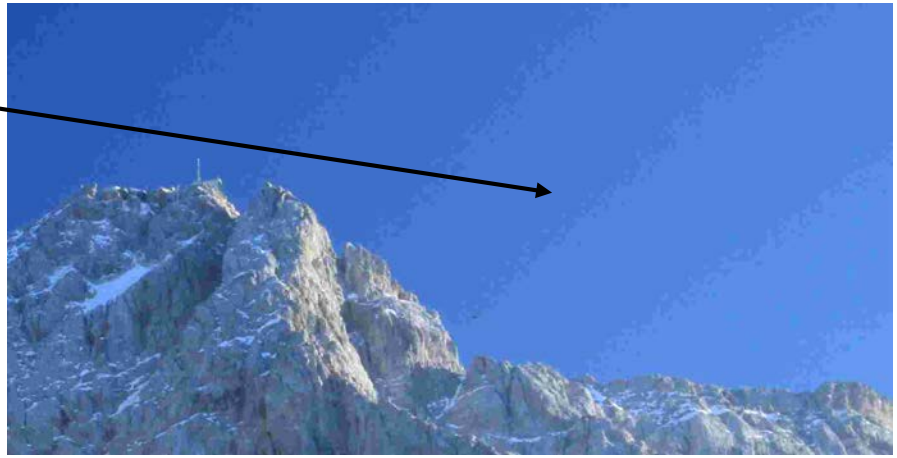
Image Files

- Images can be stored in many different file formats
  - uncompressed
  - compressed
    - lossless
    - lossy, varying quality
  - as movies

Image Files

- Example:
  - Image size 3888×2592
  - Raw data: $3888 \times 2592 \times 3 \approx 30$ MB
  - BMP-File: 30 MB
  - PNG (compressed, lossless): 13.8 MB
  - JPG (good quality): 6 MB
  - JPG (low quality): 200 kB

wikimedia commons

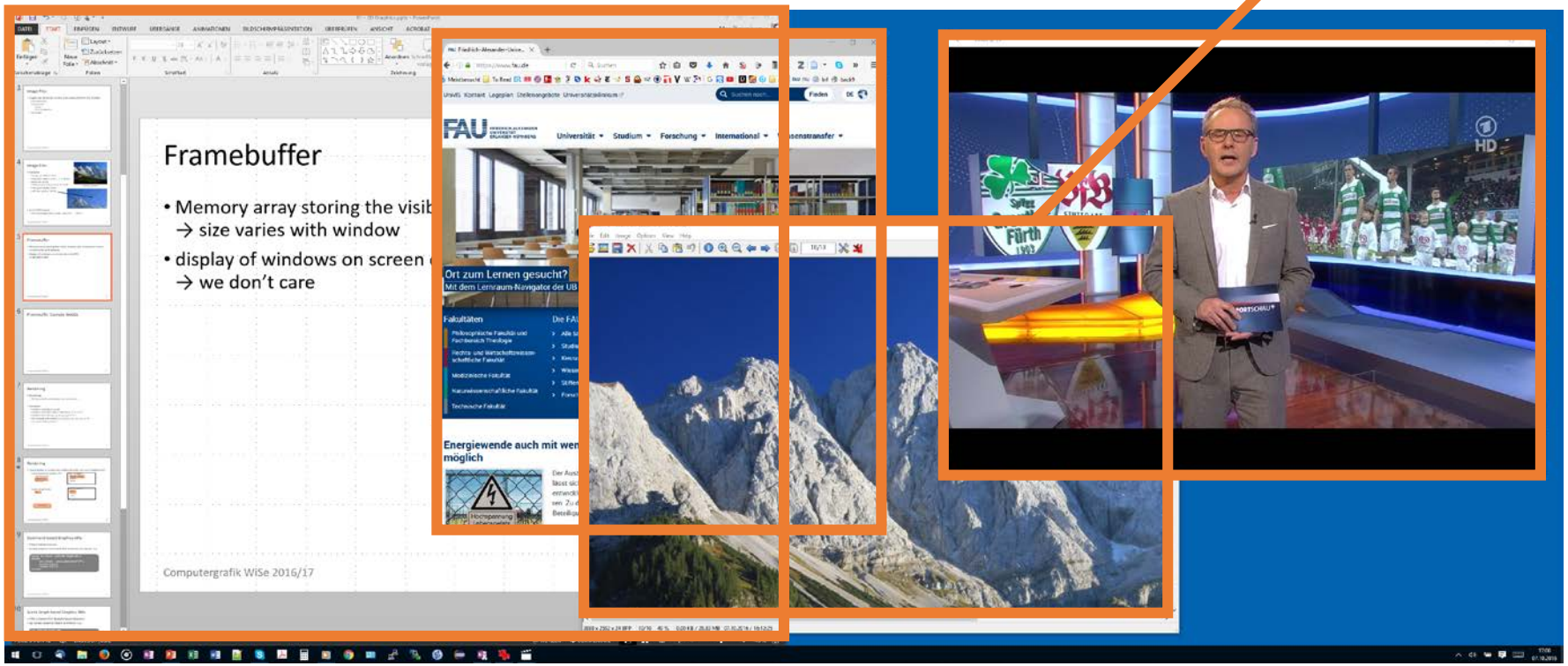


- In a FullHD movie:
  - HD-streaming 5 Mbit/s, 25fps: $5M : 8 : 25 = 25$ kB !

Framebuffer

- Memory array storing the visible content *of a window* on screen
  → size varies with window
- display of windows on screen done by GPU
  → we don't care

(at least) one framebuffer for each window



Framebuffer Example

- Directly filling the frame buffer

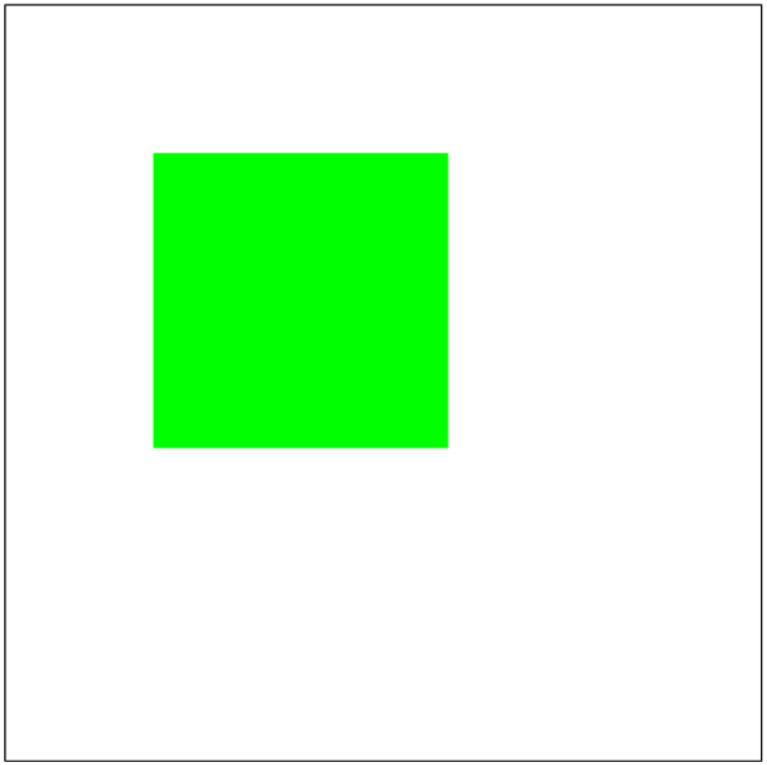
Framebuffer Example

- Examples: a green square

Framebuffer

```
var fb = getFrameBuffer();
for (var y = 100; y < 300; y++)
    for (var x = 100; x < 300; x++) {
        var i = x + 400*y;
        fb[4*i] = 0;
        fb[4*i+1] = 255;
        fb[4*i+2] = 0;
        fb[4*i+3] = 255;
    }
```

show it



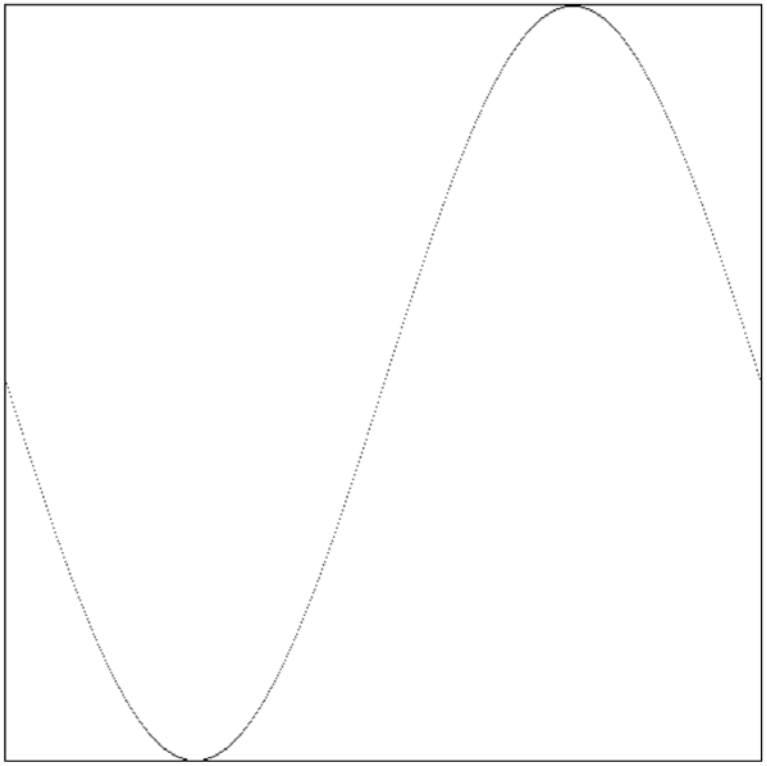
Framebuffer Example

- Examples: a sine curve

Framebuffer

```
1  var fb = getFrameBuffer();
2  for (var x = 0; x < 512; x++) {
3      var y = Math.round(Math.sin(x/256*Math.PI)*256+2
4      var i = x + 512*y;
5      fb[4*i] = 0;
6      fb[4*i+1] = 0;
7      fb[4*i+2] = 0;
8      fb[4*i+3] = 255;
9  }
10
```

show it



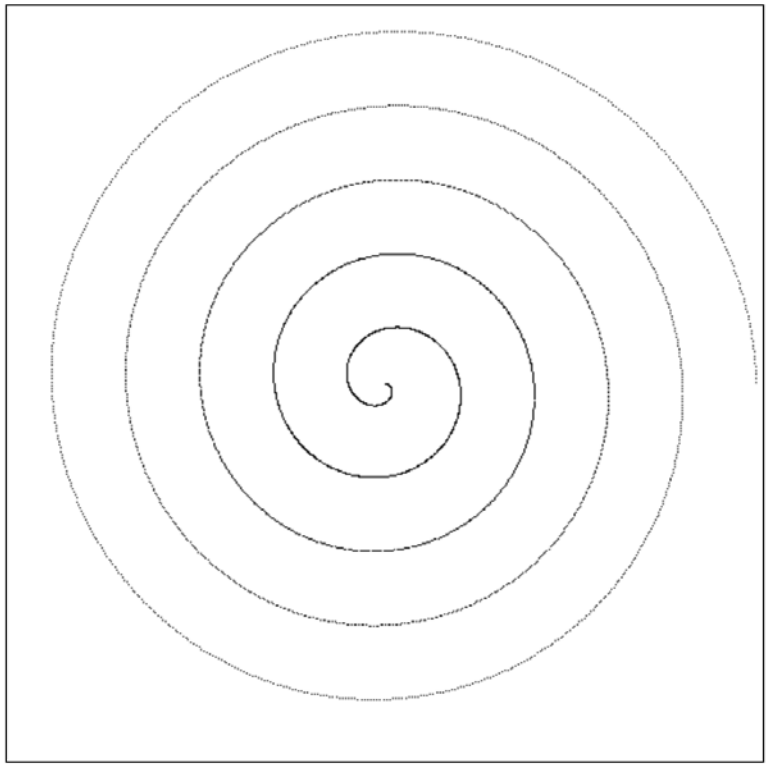
Framebuffer Example

- Examples: a spiral

Framebuffer

```
1  var fb = getFrameBuffer();
2  for (var phi = 0; phi < 10*Math.PI; phi += 0.01) {
3      var x = 256 + Math.cos(phi)*phi*8;
4      var y = 256 + Math.sin(phi)*phi*8
5      var i = Math.round(x) + 512*Math.round(y);
6      fb[4*i] = 0;
7      fb[4*i+1] = 0;
8      fb[4*i+2] = 0;
9      fb[4*i+3] = 255;
10 }
11
```

show it



Rendering

- Rendering:
  - fill frame buffer with shapes, text, 3D-content, …

- Examples:
  - render a rectangle → simple
  - render a circle with radius r and center (x, y) → ???
  - render a line from (x_1, y_1) to (x_2, y_2) → ???
  - fill a triangle with vertices (x_1, y_1), (x_2, y_2), (x_3, y_3) → ???
  - → section “Rasterization”

Rendering

- The previous example was a prank
- Frame buffer is usually not written directly, but via a Graphics API
  - Command-based graphics APIs
    - JAVA Graphics2D
    - HTML5 Canvas
    - PostScript
    - …
  - Scene Graph based
    - SVG
    - …

With 3D support:
- OpenGL / WebGL
- DirectX
- Vulcan
- …

With 3D support:
- X3D
- Unreal
- Unity
- …

in this lecture

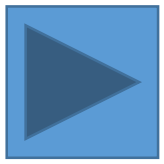
Command-based Graphics APIs

- HTML5-Element Canvas
- accepts graphics commands that draw into this canvas, e.g.

```
<canvas id="canvas" width=200 height=200 />
<script>
        var context = canvas.getContext("2d");
        context.fillStyle = "#00ff00";
        context.fillRect(100,100,300,300);
</script>
```

- for more information see:
  https://developer.mozilla.org/de/docs/Web/Guide/HTML/Canvas_Tutorial

- Examples:



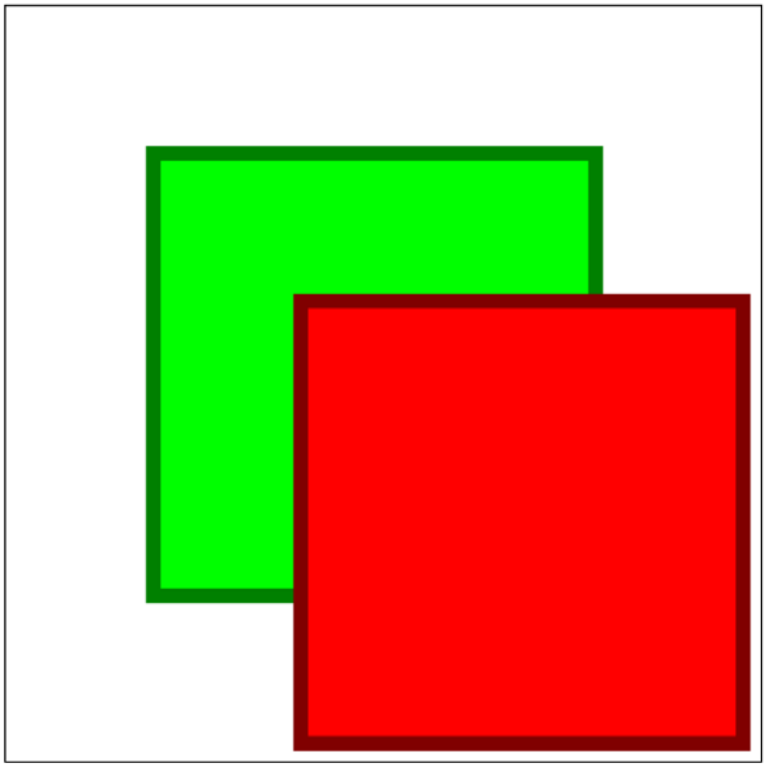
Command-based Graphics APIs

- Examples

HTML Canvas

```
1  var context = canvas.getContext("2d");
2  context.clearRect(0,0,400,400);
3
4  context.strokeStyle = "#008000";
5  context.lineWidth = 10;
6  context.fillStyle = "#00ff00";
7  context.fillRect(100,100,300,300);
8  context.strokeRect(100,100,300,300);
9
10 context.strokeStyle = "#800000";
11 context.lineWidth = 10;
12 context.fillStyle = "#ff0000";
13 context.fillRect(200,200,300,300);
14 context.strokeRect(200,200,300,300);
15
```

show it



Primitives

- Graphics are composed of *primitives* such as
  - lines
  - rectangles
  - circles / ellipses
  - triangles
  - polygons
  - curves
  - paths
- Each primitive has attributes such as
  - fill color
  - boundary color
  - line / boundary width
  - stipple pattern
  - …

Primitives - Lines

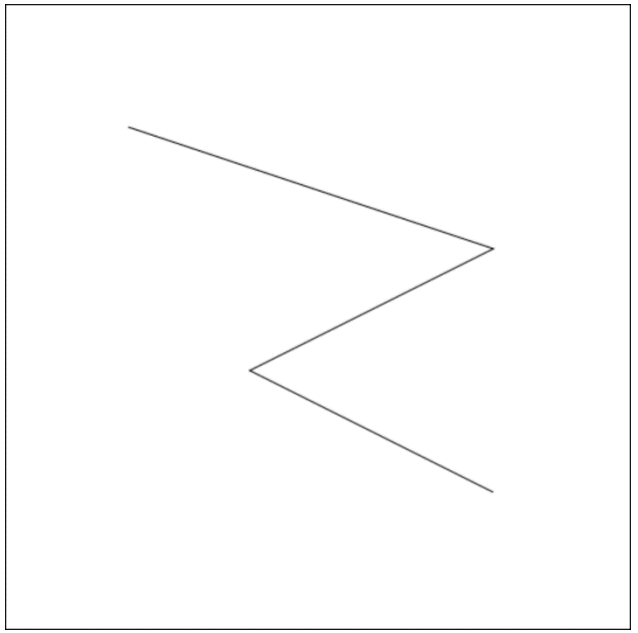
- Mathematically, a line has zero width
  → invisible
- To render a line, we have to define a **line width**
  → usually “one pixel”
- Additional attributes:
  - color
  - line caps (shape of line ends)
  - line dash (stipple patterns)

- → for more information see https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Applying_styles_and_colors

Primitives - Paths

- A path is a set of (joined) lines

```
var context = canvas1.getContext("2d");
context.clearRect(0,0,512,512);
context.beginPath();
context.moveTo(100,100);
context.lineTo(400,200);
context.lineTo(200,300);
context.lineTo(400,400);
context.stroke();
```



Primitives - Paths

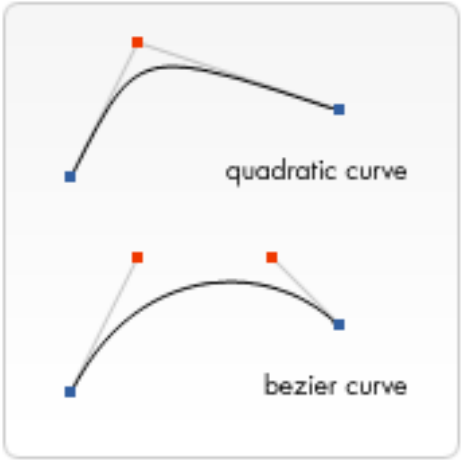
- Paths can also contain circular (or elliptical) arcs

```
var ctx = canvas2.getContext("2d");
ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
ctx.moveTo(110,75);
ctx.arc(75,75,35,0,Math.PI,false);  // Mouth (clockwise)
ctx.moveTo(65,65);
ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
ctx.moveTo(95,65);
ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
ctx.stroke();
```

Primitives - Paths

- Paths can also contain Bezier curves

```
ctx.beginPath();
ctx.moveTo(75,25);
ctx.quadraticCurveTo(25,25,25,62.5);
ctx.quadraticCurveTo(25,100,50,100);
ctx.quadraticCurveTo(50,120,30,125);
ctx.quadraticCurveTo(60,120,65,100);
ctx.quadraticCurveTo(125,100,125,62.5);
ctx.quadraticCurveTo(125,25,75,25);
ctx.stroke();
```



Primitives - Paths

- A path can also be filled

HTML Canvas

```
1  var ctx = canvas.getContext("2d");
2  ctx.clearRect(0,0,512,512);
3
4  ctx.beginPath();
5  ctx.moveTo(75,25);
6  ctx.quadraticCurveTo(25,25,25,62.5);
7  ctx.quadraticCurveTo(25,100,50,100);
8  ctx.quadraticCurveTo(50,120,30,125);
9  ctx.quadraticCurveTo(60,120,65,100);
10 ctx.quadraticCurveTo(125,100,125,62.5);
11 ctx.quadraticCurveTo(125,25,75,25);
12 ctx.stroke();
13 ctx.fillStyle="#808080";
14 ctx.fill();
```

show it

Scene Graph based Graphics APIs

- HTML5-Element SVG (**S**calable **V**ector **G**raphics)
- can contain graphical objects as children, e.g.:

```
<svg width=200 height=200>
        <circle cx=100 cy=100 r=50 />
        <rect x=0 y=0 width=100 height=100 />
</svg>
```

- for more information see:
  https://developer.mozilla.org/de/docs/Web/SVG

- Examples: 

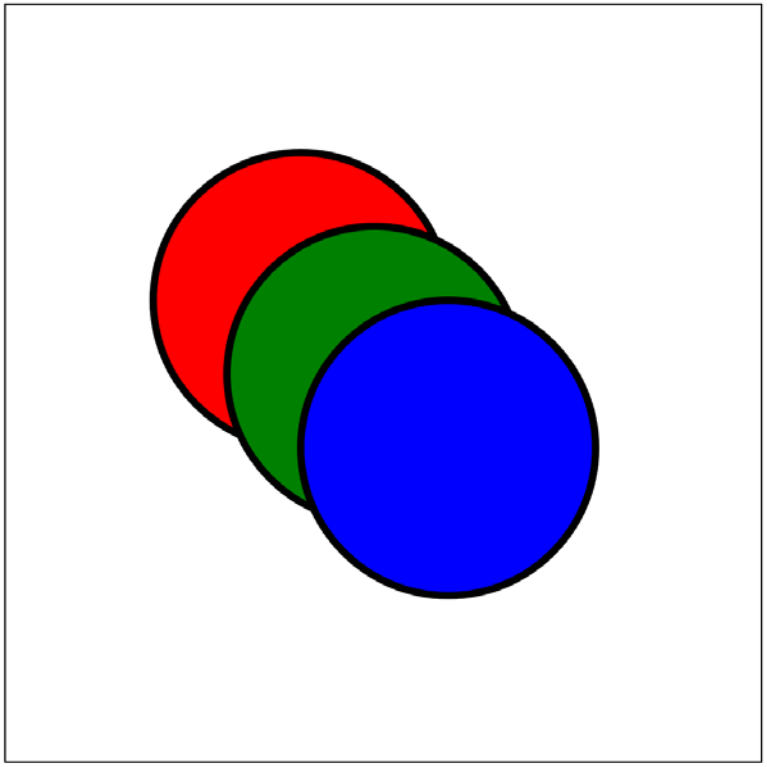
Scene Graph based Graphics APIs

- Examples:

HTML SVG

```
1  <svg id="svg">
2      <circle cx="200" cy="200" r="100"
3          stroke = "black" stroke-width=5
4          fill="red" />
5      <circle cx="250" cy="250" r="100"
6          stroke = "black" stroke-width=5
7          fill="green" />
8      <circle cx="300" cy="300" r="100"
9          stroke = "black" stroke-width=5
10         fill="blue" />
11 </svg>
12
```

show it



Scene Graph

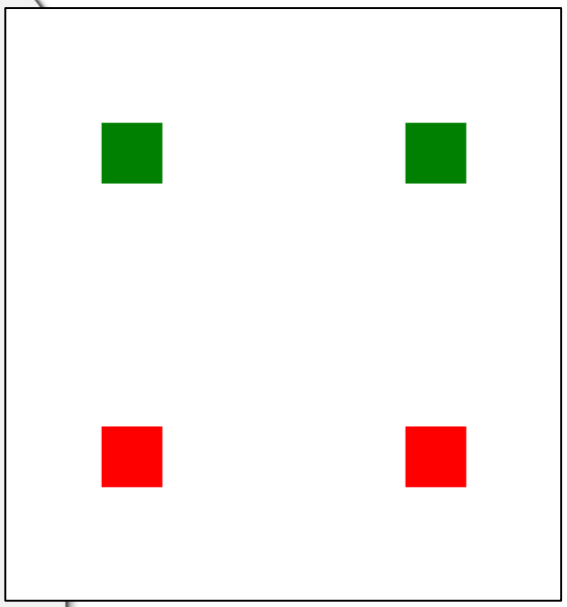
- Primitives are often arranged in a **scene graph**
- SVG allows us to group nodes using a *group node*
- allows us to apply transformations or attributes to entire groups

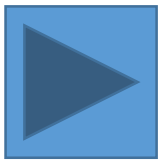
```
<svg id="svg">
    <g stroke="black" stroke-width=5>
        <circle cx="200" cy="200" r="100“ fill="red" />
        <circle cx="250" cy="250" r="100“ fill="green" />
        <circle cx="300" cy="300" r="100" fill="blue" />
    </g>
</svg>
```

Scene Graph

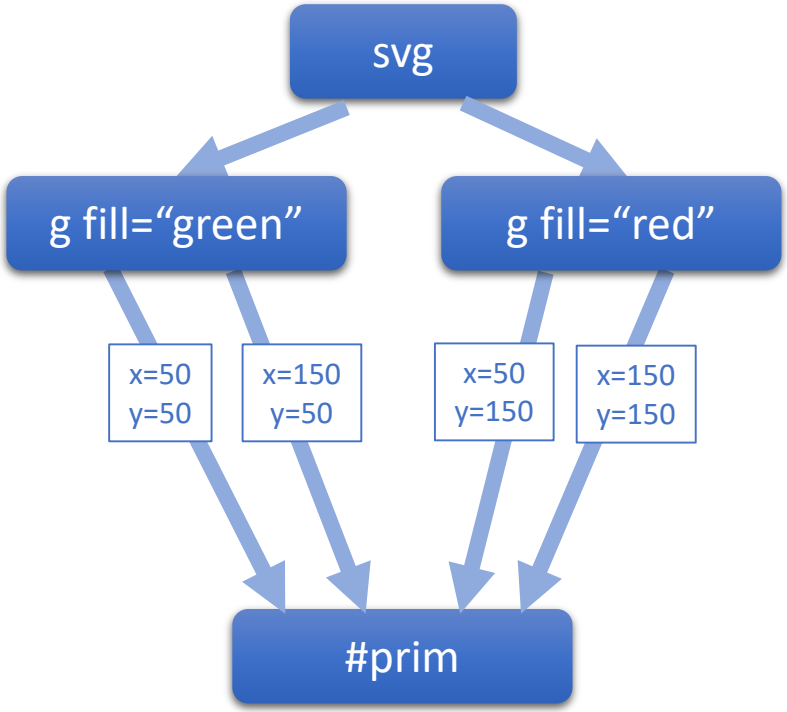
- Such a hierarchy is a **tree**, right? Why then “Scene **Graph**”?
  → group nodes can be referenced multiple times
  → graph instead of tree → “instancing”

```
<svg id="svg">
    <defs>
        <g id="prim">
            <rect width="20" height="20“ />
        </g>
    </defs>
    <g fill="green">
        <use xlink:href="#prim" x="50" y="50” />
        <use xlink:href="#prim" x="150" y="50“ />
    </g>
    <g fill="red">
        <use xlink:href="#prim" x="50" y="150“ />
        <use xlink:href="#prim" x="150" y="150“ />
    </g>
</svg>
```





Scene Graph




```
<svg id="svg">
    <defs>…</defs>
    <g fill="green">
        <use xlink:href="#prim" x="50"
        <use xlink:href="#prim" x="150
    </g>
    <g fill="red">
        <use xlink:href="#prim" x="50"
        <use xlink:href="#prim" x="150
    </g>
</svg>
```

Next lectures …

- #02: Colors
- #03: Rasterization of lines and polygons

ToDos

- register in EST
- register in studon
- get access for Huber-CIP
- download exercises
- handin exercises by next Monday