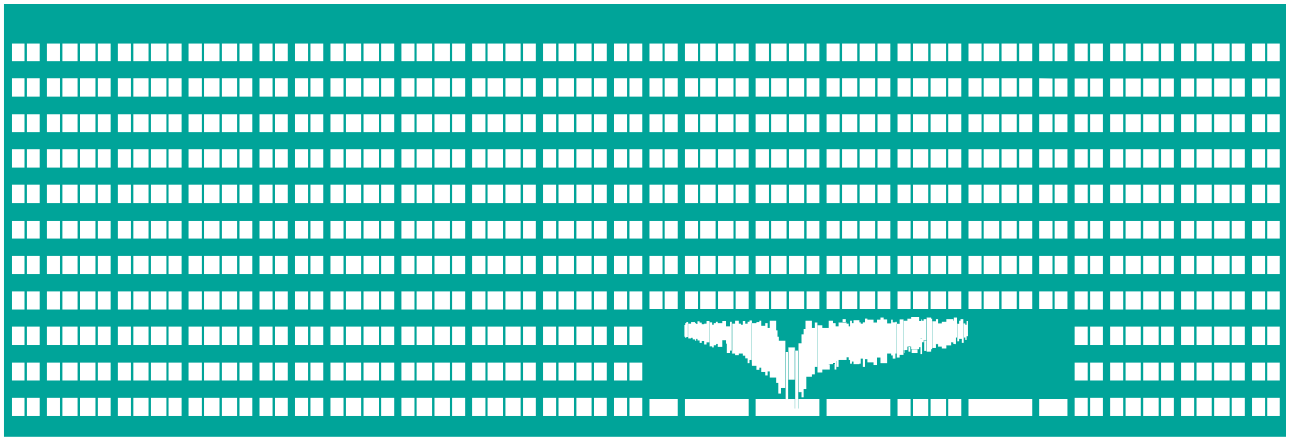
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Low-level GUI
2D Games



TAMZ 1
Lecture 6

Low-level GUI

- Every mobile platform needs a way to draw graphical primitives in (at least) 2D directly to the display/some display area and process low-level input events.
- Before HTML5, one had to use Flash to draw in web browsers, but HTML5 brings several features to do so
- 3 basic ways how to draw graphical objects on mobile platforms:
  - 2D raster graphics (result is a bitmap) – **Canvas**
  - 2D vector graphics – (Inline) **SVG**
  - 3D vector graphics rendered to Canvas – OpenGL
- We need a way how to capture (at least) keyboard and pointing device events – key pressed, released, repeated (?), mouse/touch events

Low-level GUI in HTML5

- 2D raster graphics – HTML5 Canvas
  - Supported on all current mobile platforms
  - Resolution-dependent result
  - On-the-fly drawing via JavaScript
  - events registered on predecessor; only basic text rendering
  - Well-suited for games (even graphic-intensive)
  - Several game frameworks are available
- 2D vector graphics – HTML5 inline SVG
  - Works on current mobile platforms (Android 3.0+, iOS 5.0+)
  - Resolution-independent, XML-based, has SVG DOM
    - DOM → slow rendering for complex shapes
  - Suitable for large rendering areas, unsuitable for games
  - Supports events
- 3D canvas graphics – WebGL
  - Limited support in current default mobile browsers
  - Uses OpenGL/OpenGL ES to render content into canvas

Basic low-level events in JS

- JavaScript offers 3 basic keyboard events we may process on the whole document when accessing low-level keypresses
  - keypress – keyboard button pressed (or repeated based on OS support, but it may not work e.g. for arrow keys)
  - keyup – keyboard button released, polling possible
  - keydown – keyboard button pressed, polling possible
- Mouse events (already discussed)
  - mousedown – pushed
  - mousemove – movement to new coordinates at any time (with or without pressing), event indicates state of buttons
  - mouseup – released button
  - click/tap – clicked somewhere in GUI.
  - mousecancel – cancelled, leaving touch area

Low-level touchscreen events

- On iOS and Android, we have a set of 4 touch events
  - touchstart – we started to touch the screen
  - touchmove – we moved the touch point over the screen
  - touchend or mouseup events
  - touchcancel – touch event cancelled by browser, e.g. because we are leaving touch area or touching more points than the screen supports.
  - Touch events provide following event properties:
    - **Touchlists**: touches, targetTouches, changedTouches
      - They have length property and item(index) method to provide **Touch** object with touch details:
        - Each touch provides pageX/Y, screenX/Y, clientX/Y, target and identifier (identification number for touch point through all following events) attributes
    - The ctrlKey/altKey/metaKey/shiftKey booleans provide status of these modifiers (esp. in desktop browsers)
- ~~On WP in IE10+ we have ms-pointer* events instead~~

Touch & pointer events : http://www.w3.org/TR/touch-events/ http://www.w3.org/TR/pointerevents/

Polling of input events

- Instead of reacting to individual keypresses and (v)clicks, we may set/unset flags based on key state in given time period (so we may combine e.g. UP and LEFT to create diagonal movement).
- We will need a periodically-executed function and flags to record key states → setTimeout() or setInterval() and appropriate time (e.g. 30 to 50 ms)
  - Makes sense on devices with hardware keyboard (at least direction keys) or gamepad, the rest will probably not be able to deliver events fast enough.
- Example of one way of event capture for individual keyboard and vmouse events (+arrow key polling) may be found at http://homel.vsb.cz/~mor03/TAMZ/low_level.html
  - We could easily poll for mouse button presses as well.

HTML5 2D Canvas

See e.g.: http://www.w3schools.com/html/html5_canvas.asp
http://www.html5canvastutorials.com
Specification: http://www.w3.org/html/wg/drafts/2dcontext/master/

Basic 2D Canvas Drawing Features

- Drawing primitives:
  - rectangles, arcs, lines (+width, joins & caps line styles)
  - Bezier and quadratic curves
  - fill/stroke object (i.e. fill/draw), clear rectangle
- Paths with the possibility of closing & filling do the most of the work to provide complex shapes
- Clipping support
- Scaling, rotating, translating and other transformations
- Support for images, pixel operations
- Support for web fonts
- Support for color, multicolor & gradient fills, shadows
- It is possible to save and restore drawing state of context
- Hit regions for user interaction have been defined in specification, but no support exist yet.

Canvas element in DOM

- HTML: <**canvas** id="myCID" **width**="200" **height**="100">
  - Fallback content can be provided inside of canvas element
  - Two different Canvas “sizes”
    - Canvas drawing area set by width and height
      - Default: 300px wide and 150 px high
    - Logical Canvas dimension: CSS width/height
      - Default: drawing area dimensions
- JavaScript access to existing Canvas Element
  - var c1 = document.getElementById("myCID");
  - var c2 = $("#myCID").get(0);
- Once we have the Canvas element in DOM, we need to access its drawing context (2D, 3D or webgl)
  - var context1 = c1.getContext("2d");
  - var context2 = c2.getContext("3d"); // 3D – won't work
- We can create Canvas dynamically without showing it
  - var c3 = document.createElement('canvas');

Canvas context drawing state

Contains information needed for drawing graphics primitives:

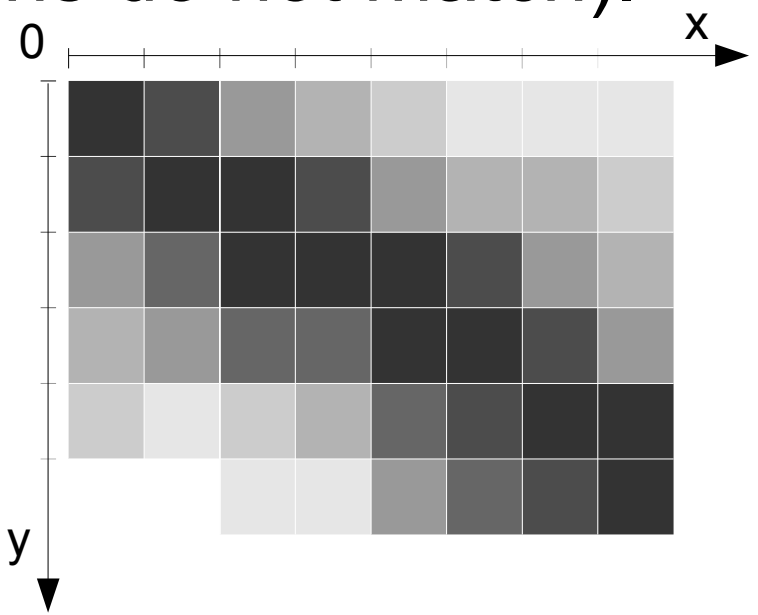
- Transformation matrix (legacy API, SVGMatrix in newer b.)
- Stroke and fill style (strokeStyle, fillStyle) – color, gradient (linear or radial) or pattern
- Global Alpha channel setting (globalAlpha) – 0.0 (tr.) →1.0
- Line settings (lineWidth, lineCap, lineJoin, miterLimit)
- Shadow setting (shadowOffsetX/Y, shadowBlur, shadowColor)
- Text setting (font, textAlign, textBaseline)
- Clipping region (of an image or when drawing)
- Global composite operation (globalCompositeOperation)

Saving and restoring drawing context state (all of above):

- context.save() – pushes context state to the stack
- context.restore() – pops context state from the stack

Coordinate system

- The coordinate system works the same way as the coordinates in the whole HTML document, relative to Canvas element (0, 0) → top-left corner pixel center.
- The GUI output may use anti-aliasing or multiple pixels on high-resolution screens (when CSS and canvas dimensions do not match).



Color Model

- The 24-bit color model consistent with CSS colors has eight bits for each of red, green, and blue, 8 bits in 32-bit integer are unused. For example #**ff**8**0**0 is orange. However, when we specify the missing 8 bits, we are requesting RGBA representation with alpha channel #RrGgBbAa (Aa=00 – transparent, Aa=ff – opaque)
  - We can use other notations: #RGBA #RGB, e.g. **#f80**, rgb(255,128, 0), rgba(rd, gd, bd, ad), hsl, hsla, gray, …
  - Default background color is in most cases “transparent black, i.e. #00000000 or rgba(0, 0, 0, 0).
  - We can adjust existing color or its component, e.g. rgba(+20, -10, 50%, *5), whiteness(20%), hue(+5), …
  - *Note: Setting alpha channel component in colors does not work in many browsers, but context.globalAlpha does.*
- Not all devices support full 24 bits of color and map the color requested by the application → color available
  - As a result, we may consider using “Web-safe colors”

For more information, see: http://dev.w3.org/csswg/css-color/

Line styles

- We can specify basic line settings one would expect in context object
  - context.lineWidth – line width (finite, greater than 0)
  - context.lineCap – line cap style (butt, round, square)

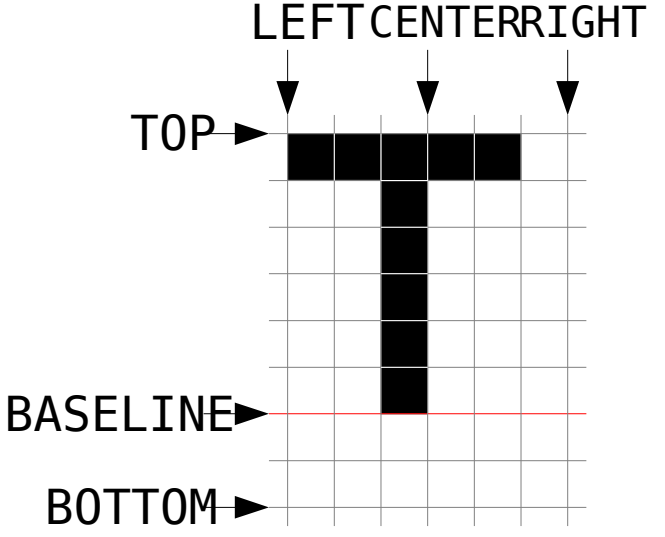
  - context.lineJoin – line join style (bevel, round, miter)

  - context.miterLimit – maximum length to cut the arrow point in miter line joins (distance between inner and outer corner of line meet), otherwise bevel line join is used
  - context.setLineDash(segments)/.getLineDash()
    - Dashed line definition: [5] → dash & space: 5px, [1, 2] → 1px dot & 2px space, [3, 5, 1, 5], …

    - context.lineDashOffset – dash pattern offset (same units)

Anchor Points and Text Alignment

- Anchor points are used to minimize the amount of required computations when placing objects (typically, **top/center/bottom/baseline** and **left/center/right**)
- For text, we also define **textAlign** and **textBaseline**, which can be used to calculate the anchor point
  - context.textAlign: start/end (depends on **direction,** which can be inherited, ltr or rtl), left, right, center
  - context.textBaseline: top, hanging, middle, alphabetic, ideographic, bottom



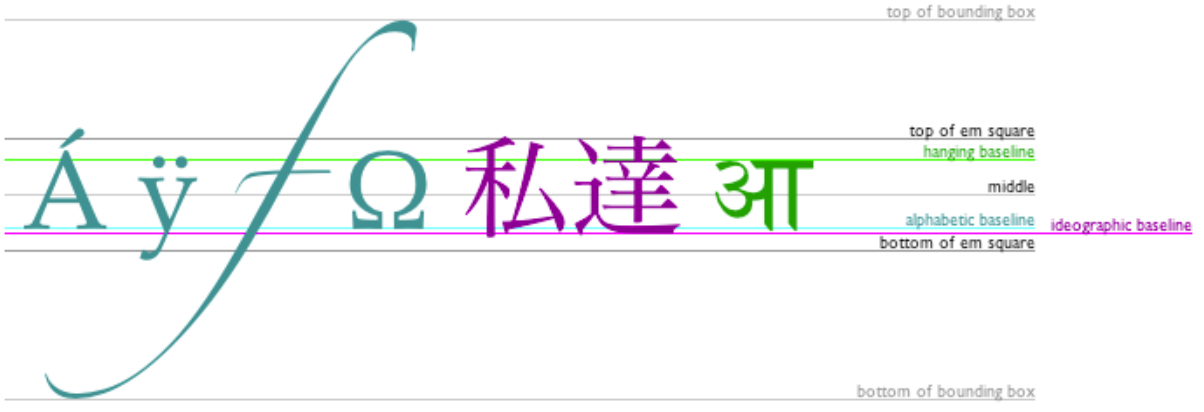


Image source: http://www.w3.org/TR/2dcontext/

Text Fonts in Canvas

The font is set with context.font, standard font definitions (know from CSS) may be used:

- We should use only vector fonts, scaling would make the result look ugly.
- Typically, we set family, size, and style.
  - Many different font families (e.g. Helvetica or Verdana)
    - Web fonts may be loaded (only on some mobile platforms)
  - Generic families: serif, sans-serif, *cursive*, fantasy, monospace
    - For generic families, it is up to the device to select a font that most closely matches the requested attributes (and maybe the font will not fit correctly).
  - text-style: normal, *italic*, *oblique* bold fonts and horizontal stretch: controlled by font-weight (400 **600**), font-stretch
  - font-size: size specifier (150%, 15px, …), adjustment or absolute: xx-small, x-small, small, medium, large, x-large, xx-large
  - Example: context.font='*italic* **600** 12px Droid Sans, sans-serif';

Coordinate transformations (2D)

- Following methods modify the transformation matrix, they are considered a legacy API, but can be still used:
  - Scaling: context.scale(scale_x, scale_y) – sets scale factors
  - Translation: context.translate(dx, dy) – adds dx and dy to current translation (negative values are possible)
  - Rotation: context.rotate(angle) – rotates clockwise by angle (in radians)
  - The transformations must be performed in reverse order.
  - Matrix manipulation: context.transform(a, b, c, d, dx, dy)
    - Multiplies current transformation matrix by 3x3 matrix based on a-f coefficients
  - context.setTransform(a, b, c, d, dx, dy) – sets the transformation matrix to supplied coefficients
- In some browsers you may use context.currentTransform to get/set transformation SVGMatrix object and use its set of methods to modify the matrix.

a	b	dx
c	d	dy
0	0	1

Fill and stroke styles

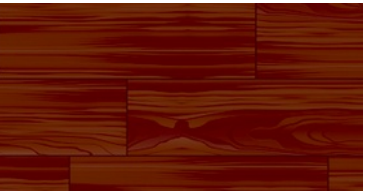
Attributes context.fillStyle, .strokeStyle may be set to:

- String containing color, e.g. #7f4f10, rgba(127,79,16,127)
  - By default, both attributes are set to #000000 (black)
- *CanvasGradient* object (gradient):
  - var grad1 = context.createLinearGradient(x0, y0, x1, y1);
    - Linear gradient along the line [x0,y0] → [x1, y1]

  - grad2 = context.createRadialGradient(x0, y0, r0, x1, y1, r1);
    - Radial gradient starting with circle [x0, y0] with radius r0 and ending with circle [x1, y1] with radius r1

  - grad1.addColorStop(offset, color);
    - Sets gradient color on given offset (between 0.0 and 1.0)

- *CanvasPattern* object (image pattern)
  - var pattern = context.createPattern(image, repetition);
    - Image: HTMLImageElement, HTMLCanvasElement, or HTMLVideoElement.
    - Repetition: repeat (both x & y), repeat-x (horizontal only), repeat-y (vertical only), and no-repeat (neither)



Casting shadows

- Each drawn object may cast a shadow controlled by following settings:
  - context.shadowColor - color of the shadow to cast.
  - context.shadowOffsetX - horizontal shadow offset
  - context.shadowOffsetY - vertical shadow offset
  - context.shadowBlur - level of blur applied to shadows. It is NOT the number of pixels.
- This part of 2D Canvas API may be subject of change in the future.

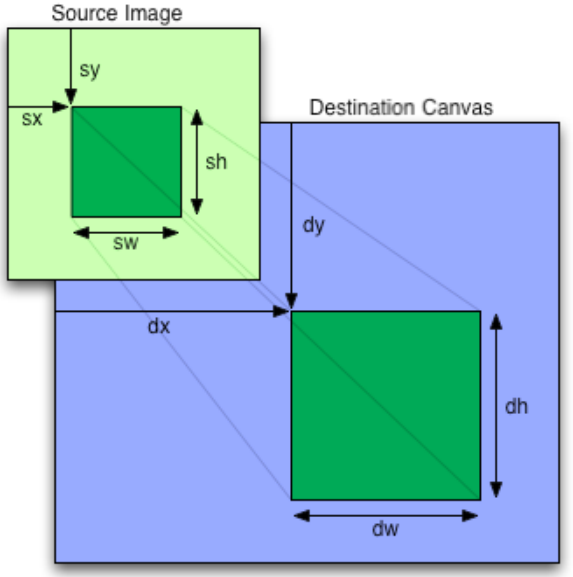
Placing images on Canvas

Draws image (or another canvas, video) on Canvas

- context.drawImage(image, dx, dy)
- context.drawImage(image, dx, dy, dw, dh)
- context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)

The arguments are interpreted as follows:

- dx, dy – image position on canvas
- dw, dh – destination size on canvas
- sx, sy – position in source image
- sw, sh – size of cut from original image



Can be used for double buffering and larger drawn area with a smaller viewport

Image source: http://www.w3.org/TR/2dcontext/

Compositing two images

- Set by context.globalCompositeOperation
- Composites image areas based on the operation
- Permitted modes for A (source) and B (destination):
  - source-atop, source-in, source-out, source-over (default)
  - destination-atop, destination-in, destination-out, destination-over
  - lighter – A+B
  - copy – A (B is ignored). Display the source image instead of the destination.
  - xor – Exclusive OR of A and B areas
  - *vendor-operation*

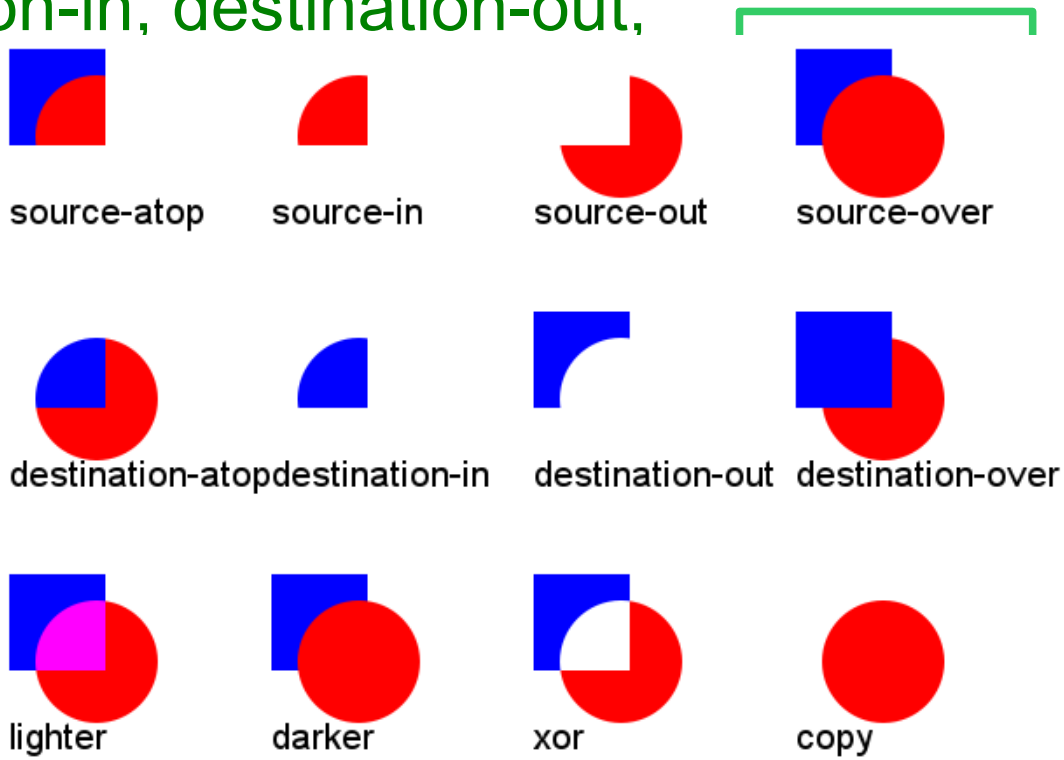


Image: http://www.html5canvastutorials.com/advanced/html5-canvas-global-composite-operations-tutorial/

Placing text on Canvas

The text can be drawn filled or as an outline

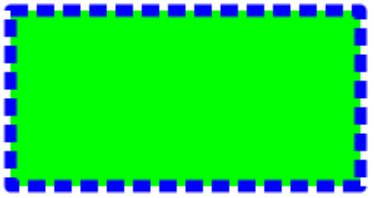
- context.fillText(text, x, y [, maxWidth]); – filled text
- context.strokeText(text, x, y [, maxWidth]); – outline text
  - maxWidth – if the text exceeds maxWidth, it will be scaled

Sometimes we need to measure the potential text width:

- var metrics = context.measureText(text);
  - returns TextMetrics object with for given text and current font set in the context
  - The actual width is then available through metrics.width
- Note that we need to supply actual text for proportional fonts, the metrics is a (#of_characters * character_width) only for `monospaced` fonts.

Rectangles in Canvas

- The rectangle is the only graphic primitive, which can be filled and cleared directly without building a path.
- Three basic methods to draw rectangles directly:
  - context.clearRect(x, y, w, h) – clears all pixels on the canvas in the rectangle to transparent black (#0000).
  - context.fillRect(x, y, w, h) – paints filled rectangle with given fill size immediately.
  - context.strokeRect(x, y, w, h) – draws a box of rectangle outline with given rectangle using current stroke style. Line styles set in context are being used.
- When combining with other objects in a path, we use context.rect(x, y, w, h) instead.



Working with paths

More complex graphical operations besides simple text and rectangle drawing are done by creating a path and filling/stroking it:

- context.beginPath() - creates/resets (empties) a path
- context.fill() – fills area defined by path's subpaths
- context.stroke() – strokes path's subpaths
  - Both fill and stroke use corresponding styles in context
- context.clip() – constrains a path by clipping region (e.g. bounding box, see general clipping later)
- context.isPointInPath(x, y) – is point [x, y] on current path or inside of the area defined by the path?
- context.drawFocusIfNeeded(element) – indicates fallback element location, if focused, draws focus outline around current path (clipping applies).

Building a path – basic shapes

- All drawing methods start with current point [x0, y0], the last point of the path, as the reference point
  - context.lineTo(x, y) – draws a straight line from current position to [x, y]. Can be used to draw polygons & triangles
  - context.moveTo(x, y) – moves to [x, y] without drawing and creates a new subpath.
  - context.rect(x, y, w, h) – rectangle as a part of the path
  - context.arc(x, y, r, startAngle, endAngle[, anticlockwise])

    - draws a circle or arc with origin [x, y] and given radius r, connected to the start of the arc from previous point by a straight line (can be avoided by context.newPath()).
    - The default drawing is from startAngle to endAngle clockwise, angles are specified in radians.
  - context.closePath() – makes the previous subpath closed (i.e. to be filled) and creates a new subpath starting at [x0, y0].

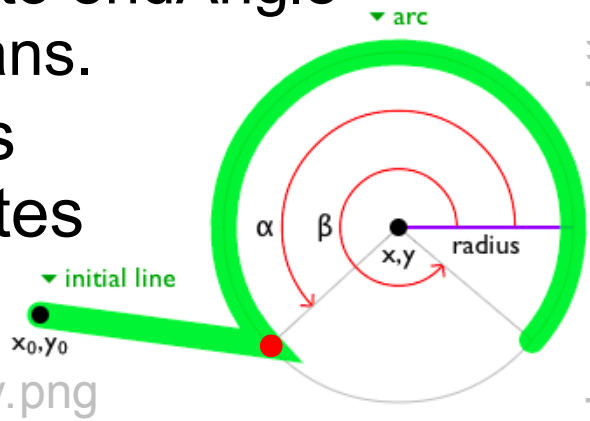


Image source: https://mdn.mozillademos.org/files/252/Canvas_smiley.png

Image source: http://www.w3.org/TR/2dcontext/

Building path – extended shapes

All three methods provide a non-linear interconnection between point [x0, y0] and [x, y]

- context.arcTo(x1, y1, x2, y2, radius) – creates line and shortest arc from actual point [x0, y0] with given radius using [x1, y1] as a reference point for tangents with [x0,y0] and [x2,y2]. The arc will end in unspecified point [x, y].
  - Note: implementation in some browsers is buggy!
  - May be used e.g. for rounded rectangles
- context.quadraticCurveTo(cpx, cpy, x, y) - draws a quadratic Bézier curve to [x, y] with control point [cpx, cpy].
- context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) - draws a cubic Bézier curve to [x, y] with control points [cp1x, cp1y] and [cp2x, cp2y].

For correct use of arcTo, see: http://www.dbp-consulting.com/tutorials/canvas/CanvasArcTo.html

String widths – automatic line break

It is possible to determine width of string typed using given font.

```
function wrapText(context, text, x, y, maxWidth, lineHeight) {
    var words = text.split(' ');
    var line = '';
    for(var n = 0; n < words.length; n++) {
      var testLine = line + words[n] + ' ';
      var metrics = context.measureText(testLine);
      var testWidth = metrics.width;
      if (testWidth > maxWidth && n > 0) {
        context.fillText(line, x, y);
        line = words[n] + ' ';
        y += lineHeight;
      } else {
        line = testLine;
      }
    }
    context.fillText(line, x, y);
}
```

All the world 's a stage, and all the men and women merely players. They have their exits and their entrances; And one man in his time plays many parts.

Source: http://html5canvastutorials.com/tutorials/html5-canvas-wrap-text-tutorial/

Clipping

- The clip is the set of pixels in the canvas context object that may be modified by graphics rendering operations.
  - The only pixels modified by graphics operations are those that lie within the clip. Pixels outside the clip are not modified by any graphics operations.
- context.clip([path]) – clips content according to specified or current path
  - .resetClip() is not present everywhere → use .save() and .restore() instead.

```
ctx.textAlign='center';
ctx.textBaseline='middle';
ctx.font='40px sans-serif';
ctx.save();
  ctx.beginPath();
  ctx.rect(leftL, topL, wL, hL);
  ctx.clip();
  ctx.fillStyle=colorL_bck;
  ctx.fillRect(0,0,width,height);
  ctx.fillStyle=colorL;
  ctx.fillText(label, text_x, text_y);
ctx.restore();
ctx.save();
  ctx.beginPath();
  ctx.rect(leftR, topR, wR, hR);
  ctx.clip();
  ctx.fillStyle=color_R;
  ctx.fillText(label, text_x, text_y);
ctx.restore();
```

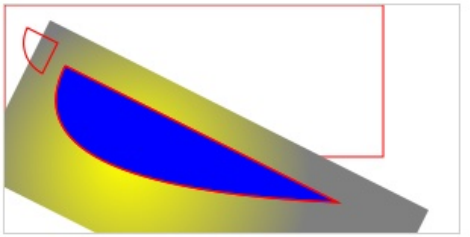
RAW access to image data

Raw data access is done through ImageData structure, which allows us to modify individual pixels. Same origin policy limitations apply. CSS dimensions are being used.

- imagedata=context.createImageData(sw, sh) – create empty image with given dimensions, filled with trans. black
- imagedata=context.createImageData(imagedata) – clones dimensions from existing buffer, fills with transparent black
- imagedata=context.getImageData(sx, sy, sw, sh) – returns pixel-based representation of canvas data from sx, sy with dimensions sw, sh.
- context.putImageData(imagedata, dx, dy [, dirtyX, dirtyY, dirtyWidth, dirtyHeight]) – puts (scaled) image data to specified canvas context
- Dimensions: imagedata.width, imagedata.height
- Actual data: imagedata.data – one-dimensional array with RGBA order and values 0-255 for each (CSS) pixel component.

HTML5 Canvas Example

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");

var grd=ctx.createRadialGradient(75,50,5,90,60,100);
grd.addColorStop(0,"yellow");
grd.addColorStop(1,"#7f7f7f");

ctx.strokeStyle="red";
ctx.strokeRect(0,0,250,100);

ctx.transform(1,0.5,-0.5,1,30,10);

ctx.fillStyle=grd; ctx.fillRect(0,0,250,100);

ctx.fillStyle="blue"; ctx.beginPath(); ctx.moveTo(20,20);
ctx.quadraticCurveTo(20,100,200,20);ctx.lineTo(20,20);

ctx.fill(); ctx.stroke();

ctx.moveTo(10,10);
ctx.arc(10,10,20,Math.PI/2, Math.PI);
ctx.closePath(); ctx.stroke();
```



HTML5 SVG support

See e.g.: http://www.w3schools.com/html/html5_svg.asp
http://www.w3schools.com/svg/svg_examples.asp
Specification: http://www.w3.org/TR/SVG/

SVG support

- SVG in HTML5
  - SVG is XML based – we can use DOM, on the other hand, Canvas is a pixel buffer (once drawn, we can't change some of the primitives)
  - Shape in SVG is represented as an object – if something changes, we re-render it when changed
  - SVG is resolution-independent vector graphics
  - Event handlers are supported
  - We can place inline SVG into HTML document
- Due to limited use of SVG in this subject, we will not discuss the SVG in detail

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" height="190">
  <polygon points="100,0 0,100 100,100 0,0"
  style="fill:yellow;stroke:red;stroke-width:5;fill-rule:evenodd;">
 </svg>
```

WebGL support
(iOS 8+, IE Mobile 11+, FF, partially Chrome on Android 40+)

See e.g.: https://developer.mozilla.org/en-US/docs/Web/WebGL
https://www.khronos.org/webgl

Examples: http://www.awwwards.com/22-experimental-webgl-demo-examples.html
http://www.html5rocks.com/en/tutorials/webgl/webgl_fundamentals/

Specification: https://www.khronos.org/registry/webgl/specs/1.0/

WebGL support

- Limited support on mobile platforms, not of much use
  - var gl = canvas.getContext("webgl");
  - gl.viewport(0,0,gl.drawingBufferWidth,gl.drawingBufferHeight);
- Based closely on OpenGL ES 2.0 API and its methods
  - context, stencils, vertices, matrices, scissors
  - textures, blends, text output
  - framebuffer and renderbuffer, pixel manipulation
  - shaders and programs, uniforms and attributes

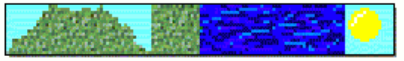
```
var canvas = document.getElementById("glcanvas");
var gl = null;
try {
    gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
}  catch(e) {}
if (gl) {
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.clear(gl.COLOR_BUFFER_BIT|gl.DEPTH_BUFFER_BIT);
}
```

HTML5 JavaScript Game Frameworks

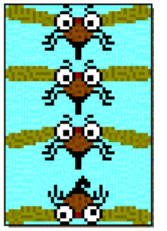
See e.g.: http://www.remcodraijer.nl/quintus/tutorial.html
Frameworks: http://html5quintus.com/ http://kineticjs.com/ https://phaser.io/

Basic game framework features

- Use existing technologies (HTML, JS, Canvas, SVG, …) but provide easier way how to write games
- We could write things which game framework provide ourselves, but it would take some time
- Typically the frameworks use well-known approaches to provide basic entities to programmers.
- Basic generalizations used in game frameworks:
  - Layers – each layer represent a different part of game (map, player, enemies, items, …)
    - TiledLayer – typically used to define maps with rectangular (sometimes hexagonal) grid. May be used both for vertical (e.g. Angry Birds) and horizontal (e.g. Civilization, PacMan, …) games.



    - Sprite – animated object which represents a player, creature, ...



  - Scene/Stage – binds layers together to provide level or some other logical part of the game.

Example game visual representation

The actual game subsystem may consist of:

- InputSystem – button and mouse input subsystem. It contains drawing methods & controls to set possible inputs
- Scene – typically a basic scene class, containing mainly the scene function and options passed to stage
- Stage/Viewport – base game object class for stage, manages sets of sprites, manipulates viewport.
- Sprite – basic sprite game object class, rendering either an asset or a frame from a sprite sheet. Width & height may be generated automatically from the supplied image assets, or set manually. Typical subtypes are:
  - MovingSprite – a sprite that adding basic Newtonian physics on each step
  - TiledLayer – tiled layer sprite used as game map
- SpriteSheet – class defining sprite parameters
- Tweens and animation sheet modules may also be used.
- Controls are used for sprite behavior, e.g. stepControls or platformerControls
- Sound effects may be included in games

- HTML5 game engine (2D games)
- JavaScript-friendly syntax (jquery-like)
- Runs on desktop and HTML5-compatible mobile devices
- Released under MIT License
- Supports Tiled layer/map/scene editor with TMX output: http://www.mapeditor.org/ or combination of images and JSON data to define the sprite animations, sheets, etc.
- Available at: http://html5quintus.com/
- Basic functions:
  - collision detection
  - sprite animations
  - tile layers
  - sound
  - input handling

For API documentation, see: http://html5quintus.com/api/

For gaming tutorial, see e.g.: http://www.remcodraijer.nl/quintus/tutorial.html

Phaser game framework

- Phaser supports following features:
  - 2D Canvas and WebGL
  - Preloading resources (images, sounds, metadata)
  - Physics for arcade, simple Axis-Aligned Bounding Box and impact physics
  - Animated Sprites and Tiled maps (with animated tiles in v3)
    - Groups of sprites are available
    - Sprite Animations by sprite sheets and some texture fmts
  - Particle system for explosions, jets, rain, …
  - Multiple Camera positions, with the possibility to follow spr.
  - Multiple inputs support (mouse, keyboard, touch screen)
  - Sounds (including new Web Audio API – mainly desktop)
  - Scaling the game for screen dimensions & full-screen
  - Plug-in support through complex plug-in system

Kinetic.js – games and graphics

Another framework for advanced image manipulations: you may add event listeners, draw shapes & images, move, scale and rotate them, use layers, detect image hits, drag&drop, …

- Canvas wrappers: Arc, Canvas, Circle, Context, Image, Line, Path, Rect, Text, Transform
- Custom graphical elements: Ellipse, Label, Ring, TextPath, RegularPolygon, Star, Tag, Wedge
- Base classes: Collection, Container, Node, Shape, Util
- Item manipulation: Filters, Easings (transitions)
- Animations: Animation, Tween
- Gaming API (offers collision detections):
  - Stage – defines the whole scene (used with other parts, too)
  - Group – grouping elements and working with whole group
  - Layer – separate GUI items with different z-index and visibility setting
  - Sprite – animated 2D sprite

For tutorial, see: http://html5canvastutorials.com/kineticjs