Assignment 2 – Geometry and GUI

UPR Mayagüez - COMP 4046 - Computer Graphics

Abstract: Implement geometric transformations and GUI interaction using the 2D HTML5 canvas.

Guidelines for assignment hand in:
- Program should be using the 2D HTML5 canvas with Javascript, unless explicitly.
Template code is provided on the Google Drive, directory Assignments/Assg2_Geometry2D/
- **One ZIP file contains all code and data necessary to run your assignment**. It is organized as follows
- A main index.html HTML file is present at top level. It provides links to all results with a short explanation.
- File organization (filenames, flat or hierarchical subfolders…) is up to you, but should be clear and meaningful.
- Make sure the example run by simply following the links, without having to modify the code. Please specify if the code should be run through a webserver as seen in class. If any additional action is required to run the results, they must be explained in a clear way in the main index.html page.
- Hand in: Ecourse https://ecourses.uprm.edu/course/view.php?id=533 before **September 13th**.

Useful references:
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial
http://www.html5canvastutorials.com/
http://glmatrix.net/

Part 1: Animation and keyboard [20]

Objective: Display animated objects and control them using the keyboard.
We want to design a game similar to asteroids, that contains a spaceship moving in a 2D toroidal space (right of the canvas is connected to the left, bottom of the canvas is connected to the top)

The template code "template-asteroid" provides the animation and interaction framework:
- A GUI shows the main parameters (x0,y0), angle, vx,vy, inSpace to be used. They are stored in the global variable params
- The refresh loop uses getAnimFrame to generate a refresh approximately every 30th second (browser dependent). Each refresh check the keyboard, update the animation and call drawAll().
- the keyboard is polled synchronously exactly once for each new refresh
The angle parameter of the GUI reacts to "left" arrow key and "right" arrow key. Current implementation creates a table that stores for each keyboard key if it is currently pressed or not and that is updated asynchronously at each keyboard event.
- drawAll() displays the wireframe of the ship, centered in (x0,y0) and rotated by angle params.angle
All drawing code is provided, you just need to implement the motion controls.

Q1) Analysis of existing code
Before answering the other questions, first analyze the code provided for yourself.
The code of drawShip() does define a "canvas transform" before drawing the triangle. By experimentation with the code and by reading the HTML5 canvas documentation, reverse engineer this code in order to explain what is does. In particular, justify why the ship is centered in (x0,y0) and has the correct angle, as illustrated in Figure 1.

Q2) Direct position control
a) Position control
When params. inSpace is false, the ship should move as follows:
Add reaction to the "up"and "down" arrows such that the spaceship moves forward and backward at velocity (vx,vy). The velocity vector should have constant norm equal to `params.velocity` and be

directed according to the forward/backward direction specified by params.angle, as illustrated in Figure 2. If the time elapsed since the last refresh is denoted `elapsed`, the ship should move by `elapsed*params.velocity` pixels.

b) The ship evolves in a 2d torus: when the spaceship passes outside of the canvas borders, teleport it to the opposite side by adding ±w to x0 or ±h to y as appropriate. (right ↔ left, top↔bottom)

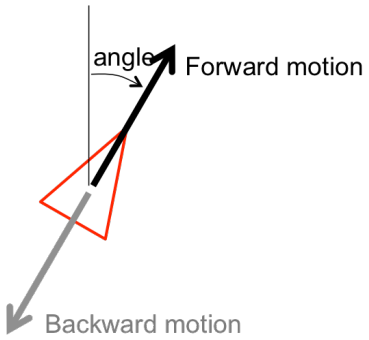

Figure 1: Specification of angle and forward/backward motion.

Q3) Velocity control
Actually, in space, we cannot control the position directly: we can only influence the velocity.
When params. inSpace is true, the ship should move as follows:
Position is updated as

```
x0 += vx * elapsed
y0 += vy * elapsed
```

The velocity (vx,vy) now takes the role of (x0,y0) is question a. It is updated by adding a constant acceleration vector (ax,ay) as:

```
vx += ax * elapsed
vy += ay * elapsed
```

This acceleration vector should be of norm params.accel, and in the direction of the forward (resp. backward) motion when the user presses the “up” key (resp. the “down” key)

Part 2: 2D Interactive Transformation [80]

Objective: We want to develop a demo that shows the effect of a 2D geometric transformation using the 2D Canvas. It follows the metaphor of a post-it note that can be translated, rotated and scaled on the display. Figure 1 shows a mockup of a typical final result.
The final application let the user:
- Draw free polylines in canvas1 (post-it content)
- Position a transformed version of canvas1 inside canvas2
- Do all interaction directly in canvas2: draw content, move the post-it around.

Note: This exercise is incremental, each question builds on the previous ones.
Submit only one code that combines your solution to all questions.
Pay attention to questions that require an explanation: provide the theoretical answers to these questions in a written report, either on paper (can be hand written) or as a separate PDF file.

Provided code:
The directory `template-postit` provides some startup template.
The HTML page has 2 canvases: input canvas and output canvas. They may have different sizes (w1,h1) and (w2,h2).

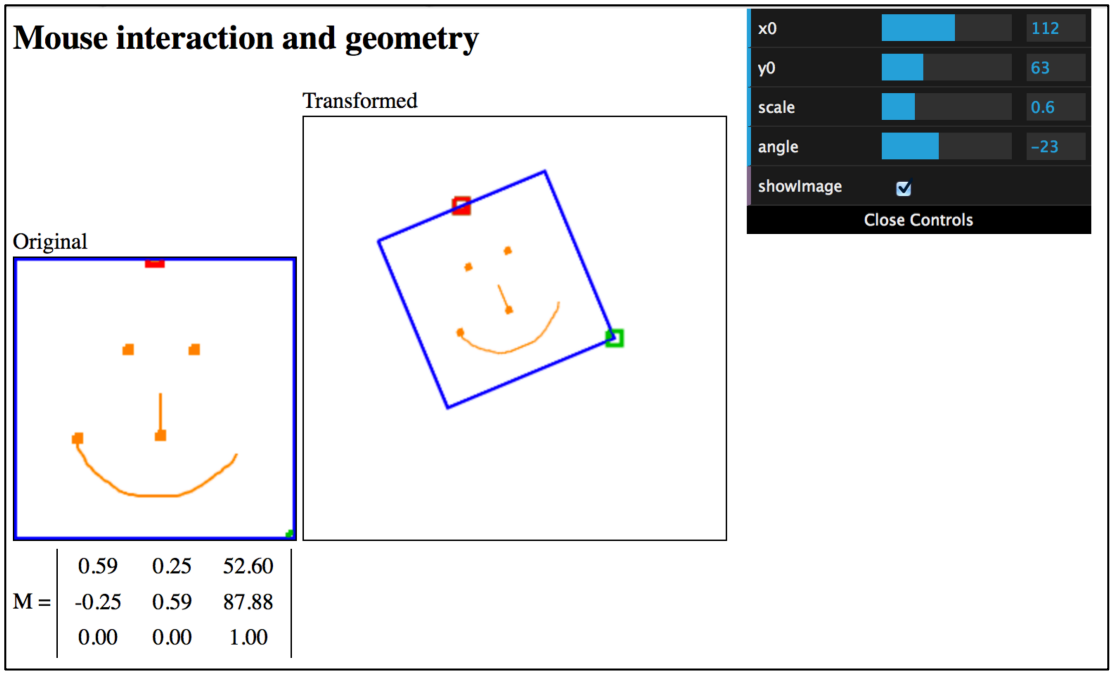

Figure 2: Mockup of the transformation demo.

The provided template already does the following:
- Provide GUI to modify parameters (x0,y0), scale and angle stored in variable `params`
- Provide drawing functions:
drawSquare(ctx, x,y, size): draw a square centered in (x,y) and of width size units.
drawShape(ctx, pts) draws the shape pts.
We define a *shape* as a flat array of interleaved 2d coordinates. For instance, the bounding box with parameters (xmin, ymin, xmax, ymax) is defined as:

```
pts = [xmin,ymin, xmax,ymin, xmax,ymax, xmin,ymax, xmin,ymin]
```

The interleaved format stores (x,y) pairs in consecutive elements:
pts[0] = x0, pts[1]=y0, pts[2]=x1, pts[3]=y1, … pts[2*i]
Note: the two previous drawing functions use the current line style of the graphical context ctx. The color, line width and other visual properties should be changed before calling drawSquare or drawShape (see examples in question Q2, Q3…)
- Provides interaction functionality to redefine (x0,y0) using the mouse (click in canvas2 close to the current (x0,y0) to start the drag operation, then dragging modifies (x0,y0))
- define a function drawAll() that should be called whenever the display is updated

Q1) Interaction: painting polylines
Use the mouse callbacks of canvas1 to enable the user to paint the content of the canvas:
- when the user clicks on the canvas, a small dot is drawn in canvas1 (this part is already done)
- when the user click, then drag on the canvas, a curve (polyline) is drawn between the successive positions of the mouse until the user releases the mouse.
Hint: The following callback code converts the mouse position from “client coordinates” (in the HTML window) to “canvas coordinates” in pixels.

```
function onMouseClick(event) {
   var rect = event.target.getBoundingClientRect()
   var xpix = event.clientX - rect.left   // top-left: (xpix,ypix)=(0,0)
   var ypix = event.clientY - rect.top
   ...
}
```

Note: the previous conversion code works if the canvas is a top-level element in the HTML page. If the coordinates appear with an offset, make sure the canvas is a top-level element.

Q2) Drawing: Shape

To help us in dealing with geometrical transforms, we will implement some helper functions.

a) Write a function `drawShape(ctx, pts)` that draws an arbitrary shape `pts` using context `ctx`. We define a *shape* `pts` as a flat array of interleaved 2d coordinates. For instance, a polyline passing through the points (x1,y1), (x2,y2) and (x3,y3) is defined as

```
pts = [x1,y1, x2,y2, x3,y3]
```

b) Use this function to draw canvas1 bounding box (use a width larger than 3 for this to be visible). For instance:

```
pts1 = [0,0, w1,0, w1,h1, 0,h1, 0,0]
ctx1.strokeStyle = 'blue'
ctx1.lineWidth = 3
drawShape(ctx1, pts1)
```

Q3) Geometry: Transform points using 3x3 matrix

We want to transform the content of canvas1 and display it onto canvas2 using translation, rotation and scale. In this question, only the bounding box shape of canvas1 is considered.

a) Write a function `transformPoints(M, pts)` that applies a transformation `M` to a shape `pts`, and returns a transformed shape `pts2`.

The *3x3 transformation matrix* M is defined as a mat3 object (using glMatrix library, see appendix)

Note: You are asked to do the computation yourself and/or by using the glMatrix utility functions, no canvas operation is involved here. See appendix for usage of glMatrix and data structure conventions.

b) Use transformPoints to display the transformation of canvas1's bounding box into canvas2

For instance, for a translation of vector (x0,y0):

```
var M = mat3.create();
mat3.identity(M)
mat3.translate(M, M, [params.x0, params.y0])
pts2 = transformPoints(M, pts1)
ctx2.strokeStyle = 'blue'
ctx2.lineWidth = 3
drawShape(ctx2, pts2)
```

c) Modify your code to follow these specifications:

- Point (w1/2, 0) is transformed into (params.x0, params.y0) (We call this point the Red Pin)
- Content is rotated by params.angle, and scaled by params.scale

Explain your reasoning and how you checked if your code meets these specifications.

Hint: You may define M either by manipulating directly the coefficients M[i], or by using the methods translate, rotate, scale.

Hint2: You may use the provided matrix display code to display the content of M on the HTML page:

```
matElem = document.getElementById('mat')
matElem.innerHTML = 'M = '+mat3.toHTML(M)
```

Q4) Drawing and geometry: Transform the image

To transform images, we will use a different approach, using "canvas transform". Once a canvas transform has been associated to a canvas, every drawing operation on this canvas will receive the transform automatically before drawing. By default, the canvas transform is the identity.
Read the HTML5 canvas tutorial for more information.

Copy the content of canvas1 onto canvas2 using "canvas transform". The function drawImage() can take canvas1 as the image content to be drawn.

```
ctx2.setTransform( ??? TODO ??? )
ctx2.drawImage(canvas1, 0,0)
ctx2.setTransform(1, 0, 0, 1, 0, 0) //Reset when done
```

The coefficients of matrix M should be passed to setTransform.
Justify the parameters of setTransform: in which order should the coefficients M[i] be used ?
Hint: Read carefully the documentation of setTransform() and of glMatrix.

Combining Q1 and Q4, you should now be able to draw a picture in canvas1 and display it in canvas2 with a transformation.

Q5) Interaction and geometry: painting in transformed space

We now want to be able to paint canvas1 directly by interacting with canvas2.
Add a callback to canvas2 so that the user can paint on canvas1 directly from the canvas2 display. You should compute the correct geometrical transformation of the mouse events to redirect all painting actions back to canvas1: no painting is done directly to canvas2.
Hint: `mat3.invert(…)` computes the inverse of a 3x3 matrix.
Note: This functionality should not interfere with the translation of (x0,y0) when clicking close to the red pin.

Bonus work: (worth max 10 points: pick one, or propose something of similar difficulty)
In part1:
Finish the asteroid game by adding at least one asteroid. Pressing Space launches a bullet in the forward motion axis that continues at constant speed until it destroys the asteroid or a predefined time has elapsed. Both asteroid and bullet follow a 'constant velocity' model where their (vx,vy) velocity vector remains constant once it has been defined.
In part2:
The user can already translate the "post-it" note by clicking and dragging the red pin at the top. Add the functionality to modify scale and angle intuitively by clicking and dragging one of the corners of the "post-it" in canvas2 (for instance the Green handle at the bottom right of the rectangle in Figure 1). Ideally, angle and scale should be updated so that the corner follows precisely the mouse position. Explain your reasoning.
Note: Scaling and Rotation can be handled independently. Scaling is easier than rotation. Math.atan2 is useful for estimating the rotation.

Technical appendix: using glMatrix

The most recent version of glMatrix can be found here http://glmatrix.net/, with its documentation. It is advised to use version 2 instead of the older version 1 used in the Learning WebGL lessons, as it is better documented and has a more consistent API. The two APIs are similar but incompatible.

The naming is straightforward:
vec2 = 2D vector, vec3 = 3D vector, vec4 = 4D vector
mat2 = 2x2 matrix, mat3 = 3x3 matrix, mat4 = 4x4 matrix

To manipulate 2D geometry in homogeneous coordinates, we therefore need vec3 and mat3.
To manipulate 3D geometry in homogeneous coordinates, we therefore need vec4 and mat4.

Allocation: Must always be done once before manipulating a matrix or vector/point

```
var M = mat3.create();
var P = vec3.create();
```

Accessing elements:
Vectors store elements as a simple array, elements are accessed directly as

```
P[0], P[1], P[2]
```

For matrices, elements are stored in *column-major order in a flat array*.
For instance, for the 3x3 matrix

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix},$$

we access the coefficients as follows:

```
M[0] = a;    M[3] = b;    M[6] = c;
M[1] = d;    M[4] = e;    M[7] = f;
M[2] = 0;    M[5] = 0;    M[8] = 1;
```

Copying content:

```
mat3.copy = function(Mout, M)
```

Copies the content of M into Mout.
Important notes related to copying/passing matrices as parameters:
- The code Mout = M, does NOT copy the content, it copies the pointer to the SAME array in memory. Modifying Mout will also modify M.
- Be careful when modifying the content of a matrix passed as a parameter to a function, as it is passed by reference, not copied, so when exiting the function, the initial matrix has been modified.
- Identify clearly for each function parameter if it is input or output. Using the same matrix as both input and output to a function is risky; make sure your code is safe if you do so.

Matrix operations:

```
mat3.identity = function(Mout)
```

Define Mout to be the identity matrix.

```
mat3.mult = function(Mout, M, M2)
```

Compute the product M*M2 and store it in Mout (notice the order of the arguments)

`mat3.translate = function(Mout, M, T)`
Compute the product M*MT and store it in Mout, where MT is the 3x3 matrix representing translation by 2D vector T. (notice the order of the arguments)
Can be used to generate a translation matrix by passing the identity matrix in M.
It is safe to use same matrix for Mout and M in this function.

`mat3.rotate = function(Mout, M, theta)`
Compute the product M*R and store it in Mout, where R is the homogeneous 3x3 rotation matrix representing 2D rotation by angle theta. (Notice the order of the arguments)
Can be used to generate a rotation matrix by passing the identity matrix in M.
It is safe to use same matrix for Mout and M in this function.

see glMatrix documentation for an exhaustive list of methods.

Matrix-vector product
Matrix-vector product is defined for mat4 and vec4 objects:
`vec4.transformMat4 = function(out, a, m).`
Compute 4-vector out = m * a, where a is a 4-vector and m a 4x4 matrix

Same function exists for vec3 and mat3. Here is example code

```
vec3.transformMat3 = function(out, a, m) {
    var x = a[0], y = a[1], z = a[2];
    out[0] = m[0] * x + m[3] * y + m[6] * z;
    out[1] = m[1] * x + m[4] * y + m[7] * z;
    out[2] = m[2] * x + m[5] * y + m[8] * z;
    return out;
};
```

Notice that since the content of **a** is copied into local variables before being used, it is safe to use same vector for **out** and **a** in this function.

Displaying matrix
For debugging, it is useful to display the content of a matrix or vector.
Simple approach: print it to the console:

```
console.log(M)
```

→ *Float32Array* [1, 0, 0, 0, 1, 0, 0, 0, 1]
Note : be careful that the coefficients are printed in column-major order !

GUI based: convert the matrix to HTML code and inject it inside the HTML page

```
mat3.toHTML = function(m) {
    var str = '<table style="border-left:solid 1pt; border-right:solid 1pt;"><tbody>'

    str += '<tr><td>'+m[0].toString()+'</td><td>'+m[3].toString()+'</td><td>'+m[6].toString()+'</td></tr>';
    str += '<tr><td>'+m[1].toString()+'</td><td>'+m[4].toString()+'</td><td>'+m[7].toString()+'</td></tr>';
    str += '<tr><td>'+m[2].toString()+'</td><td>'+m[5].toString()+'</td><td>'+m[8].toString()+'</td></tr>';

    str += '</tbody></table>'

    return str
}
```

then, in the main code:

```
matElem = document.getElementById('mat')
matElem.innerHTML = 'M = '+mat3.toHTML(M)
```

which will replace the content of the HTML tag

```
<div id='mat'></div>
```