UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

CG Basics IV – Basic Interaction in WebGL

Bachelor's Degree in Video Game Design and Development

Keyword Events

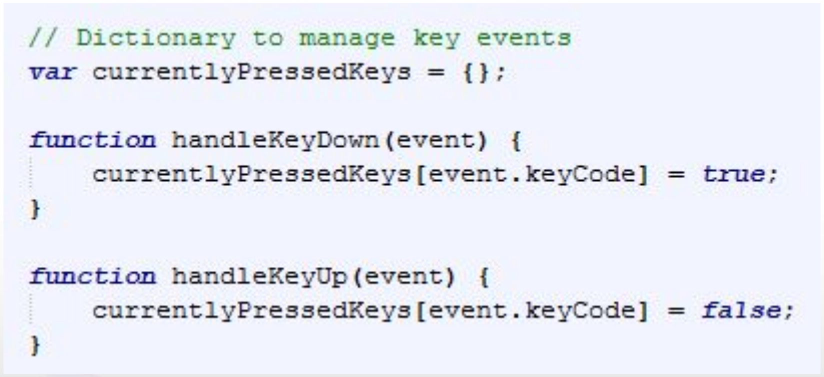
- **To handle keyboard events we must follow different steps**
  - **Receive key events in the webpage (*WebStartGL()*)**

```
document.onkeydown = handleKeyDown;
document.onkeyup = handleKeyUp;
```

  - **And associate them to the functions that will be executed when a keyword event is received**
    - **handleKeyDown is executed when a key is pressed**
    - **handleKeyUp is executed when a key is released**

Keyword Events

- To handle keyboard events we must follow different steps
  - We use a dictionary (pairs: key, value) to manage the key events
  - Whenever a key is pressed, the entry of the key in the dictionary is set to true
  - When a key is released, its entry in the dictionary is set to false

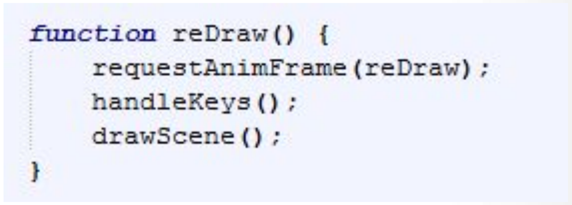


```
// Dictionary to manage key events
var currentlyPressedKeys = {};

function handleKeyDown(event) {
    currentlyPressedKeys[event.keyCode] = true;
}

function handleKeyUp(event) {
    currentlyPressedKeys[event.keyCode] = false;
}
```

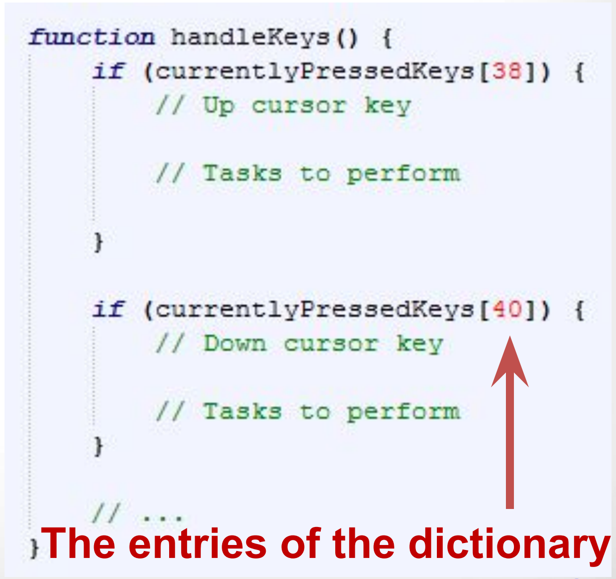
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Keyword Events

- To handle keyboard events we must follow different steps
  - The tasks to do when a certain key is pressed is defined in another function (*handleKeys()*), called in the reDraw() function
  - In this way, we assure that the scene is updated whenever a key is pressed



```
function reDraw() {
    requestAnimFrame(reDraw);
    handleKeys();
    drawScene();
}
```



```
function handleKeys() {
    if (currentlyPressedKeys[38]) {
        // Up cursor key

        // Tasks to perform

    }

    if (currentlyPressedKeys[40]) {
        // Down cursor key

        // Tasks to perform
    }

    // ...
}
```

The entries of the dictionary correspond to javascript KeyCodes

Mouse Events

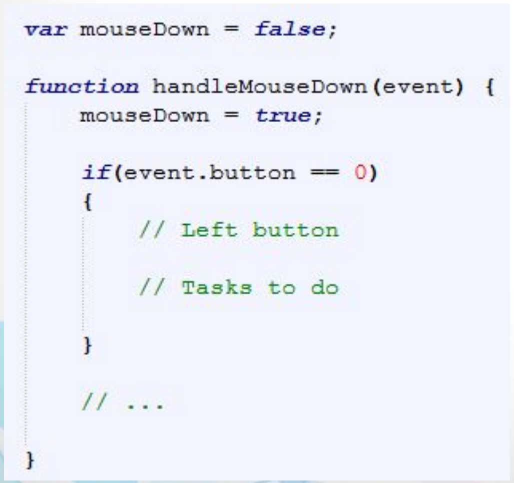
- To handle mouse events we must
  - Know when the mouse is clicked on the canvas
  - As well, when a mouse button is released on the canvas or the webpage

```
canvas.onmousedown = handleMouseDown;
document.onmouseup = handleMouseUp;
document.onmousemove = handleMouseMove;
```

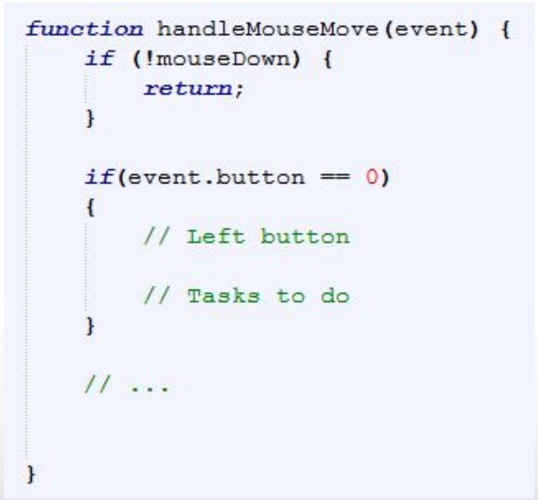
  - And associate them to the functions that will be executed when a mouse event is received
    - handleMouseDown is executed when a button is clicked
    - handleMouseUp is executed when a button is released
    - handleMouseMove is executed while dragging

Mouse Events

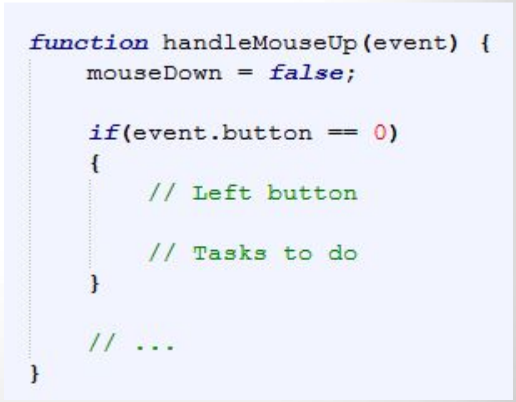
- Mouse buttons are codified as follows
  - Left button → event button 0
  - Mid button → event button 1
  - Right button → event button 2
- Example



```
var mouseDown = false;

function handleMouseDown(event) {
    mouseDown = true;

    if(event.button == 0)
    {
        // Left button

        // Tasks to do

    }

    // ...

}
```



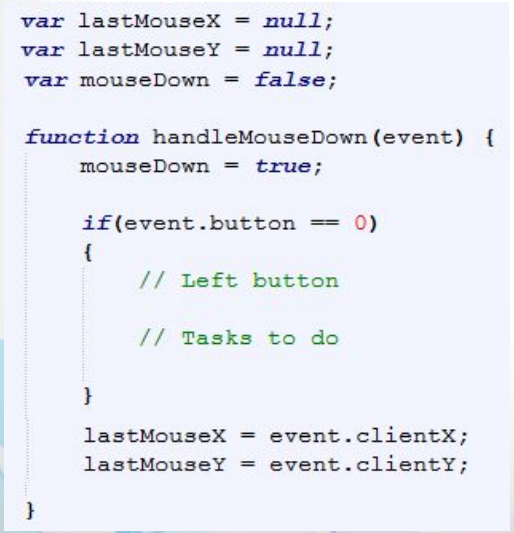
```
function handleMouseMove(event) {
    if (!mouseDown) {
        return;
    }

    if(event.button == 0)
    {
        // Left button

        // Tasks to do
    }

    // ...


}
```



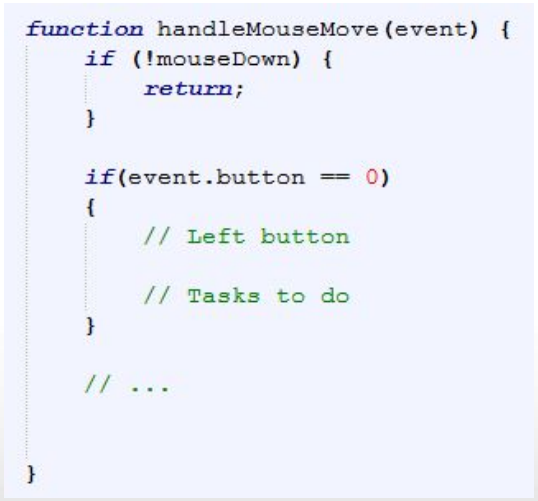
```
function handleMouseUp(event) {
    mouseDown = false;

    if(event.button == 0)
    {
        // Left button

        // Tasks to do
    }

    // ...
}
```

Mouse Events

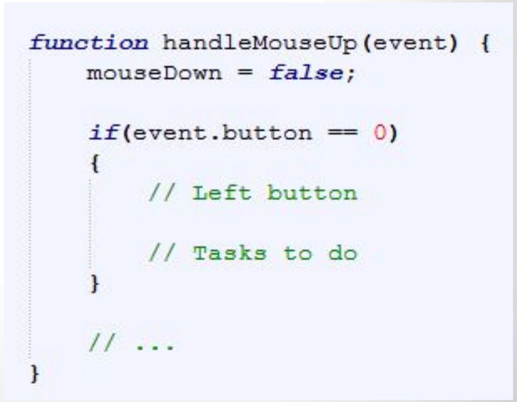
- Mouse buttons are codified as follows
  - Left button → event button 0
  - Mid button → event button 1
  - Right button → event button 2
- Example



```
var lastMouseX = null;
var lastMouseY = null;
var mouseDown = false;

function handleMouseDown(event) {
    mouseDown = true;

    if(event.button == 0)
    {
        // Left button

        // Tasks to do

    }
    lastMouseX = event.clientX;
    lastMouseY = event.clientY;
}
```



```
function handleMouseMove(event) {
    if (!mouseDown) {
        return;
    }

    if(event.button == 0)
    {
        // Left button

        // Tasks to do
    }

    // ...

}
```



```
function handleMouseUp(event) {
    mouseDown = false;

    if(event.button == 0)
    {
        // Left button

        // Tasks to do
    }

    // ...
}
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Basic Interaction

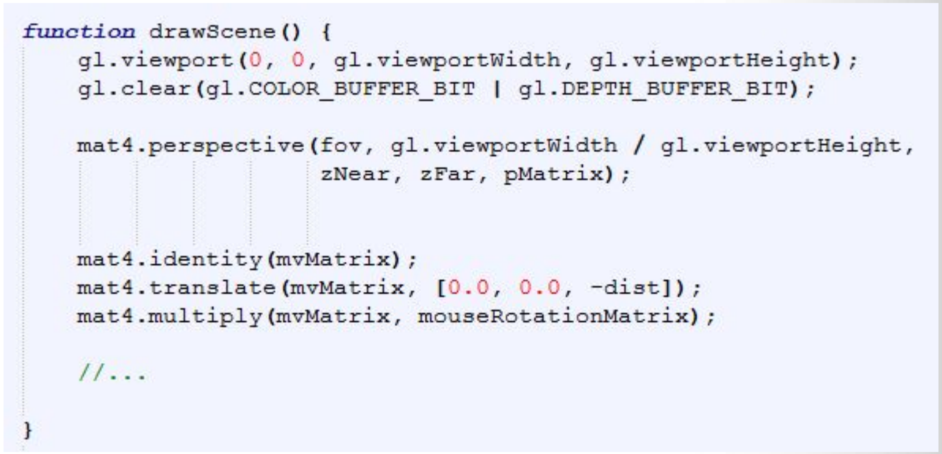
- **Scene rotation**
  - **In order to rotate the scene when interacting with the mouse, we accumulate all the rotations in a 4x4 matrix**
  - **The mouse displacement is used to specify the rotational angle**
  - **In order to accumulate the rotations correctly, we must "premultiply" each new rotation to the "accumulation matrix"**

```
var mouseRotationMatrix = mat4.create();
mat4.identity(mouseRotationMatrix);

function handleMouseMove(event) {
    if (!mouseDown) {
        return;
    }

    var newX = event.clientX;
    var newY = event.clientY;
    var deltaX = newX - lastMouseX
    var deltaY = newY - lastMouseY;

    var newRotationMatrix = mat4.create();
    mat4.identity(newRotationMatrix);
    mat4.rotate(newRotationMatrix, deg2Rad(deltaX / 10),
                [0, 1, 0]);
    mat4.rotate(newRotationMatrix, deg2Rad(deltaY / 10),
                [1, 0, 0]);
    mat4.multiply(newRotationMatrix, mouseRotationMatrix,
                  mouseRotationMatrix);

    lastMouseX = newX
    lastMouseY = newY;
}
```

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Basic Interaction

- Scene rotation
  - In the end, the computed rotation matrix must be applied to the modelview (*drawScene()*)
  - Remember that geom. transforms must be applied in a specific order!!



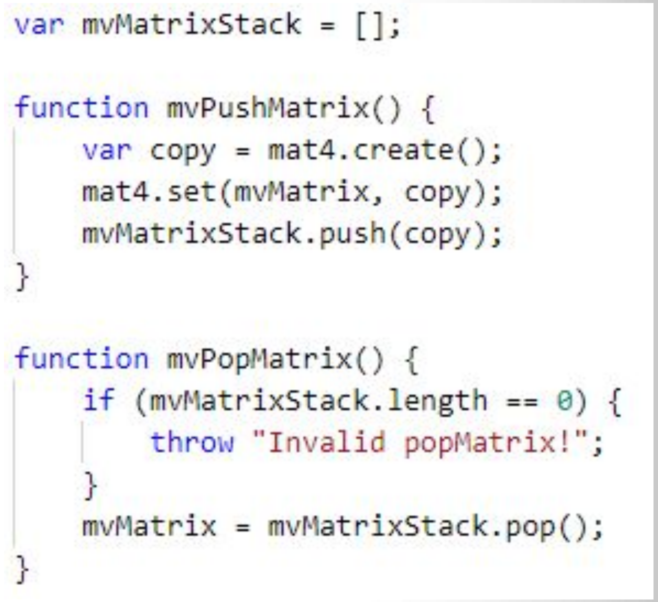
```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(fov, gl.viewportWidth / gl.viewportHeight,
                         zNear, zFar, pMatrix);


    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0.0, 0.0, -dist]);
    mat4.multiply(mvMatrix, mouseRotationMatrix);

    //...

}
```

Basic Interaction

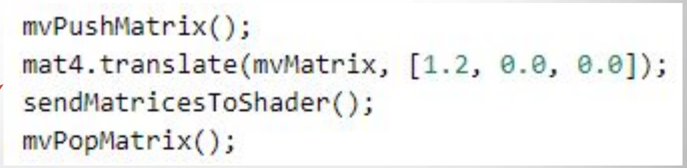
- Panning
  - Moving the camera horizontally and vertically with mouse or key events
    - Displacements in X and Y should be computed when interacting with the mouse or keyboard and applied to the scene (*drawScene()*)
    - They should be applied in the observer's coordinates system!
- Zooming
  - It can be done in two different ways (results are slightly different)
    - Varying the FOV of a perspective camera
    - Moving closer of further the scene in the observer's Z axis

Matrix stack

- Different objects, different transforms
  - Projection and modelview matrices
    - Modelview = view * model
    - View matrix is shared among all objects…
- Reset and define modelview for each object?
  - NO!
- For convenience, we can simulate glPushMatrix() / glPopMatrix()



```
var mvMatrixStack = [];

function mvPushMatrix() {
    var copy = mat4.create();
    mat4.set(mvMatrix, copy);
    mvMatrixStack.push(copy);
}

function mvPopMatrix() {
    if (mvMatrixStack.length == 0) {
        throw "Invalid popMatrix!";
    }
    mvMatrix = mvMatrixStack.pop();
}
```



```
mvPushMatrix();
mat4.translate(mvMatrix, [1.2, 0.0, 0.0]);
sendMatricesToShader();
mvPopMatrix();
```

After defining the camera transform and for each object...

Questions?

www.citm.upc.edu